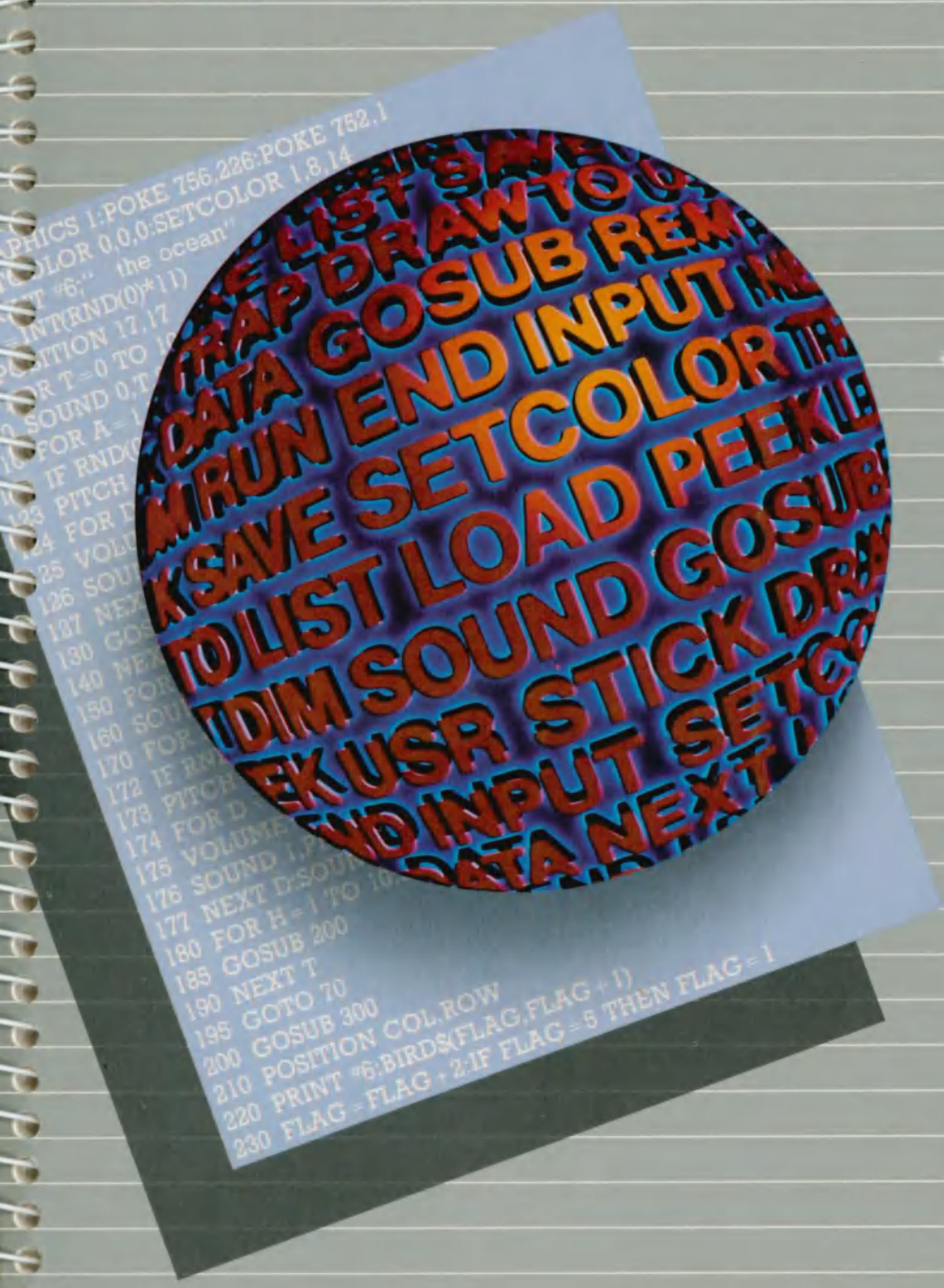
ATARI® BASIC

Reference Manual



BASIC REFERENCE MANUAL

A Warner Communications Company

Every effort has been made to ensure that this manual accurately documents the operation of the ATARI 400™, ATARI 800™, and the ATARI 1200XL™ Home Computers However, because we are constantly improving and updating our computer software and hardware, ATARI, INC is unable to guarantee the accuracy of printed material after the date of publication and disclaims liability for changes, errors, or omissions

© 1983 ATARI, INC All rights reserved
No reproduction of this manual or any portion of its contents is allowed without specific written permission of ATARI, INC

ERROR CODES

ERROR CODE	ERROR CODE MESSAGE
2	Memory Insufficient
3	Value Error
4	Too Many Variables
5	String Length Error
6	Out of Data Error
7	Number greater than 32767
8	Input Statement Error
9	Array or String **DIM** Error
10	Argument Stack Overflow
11	Floating Point Overflow/ Underflow Error
12	Line Not Found
13	No Matching **FOR** Statement
14	Line Too Long Error
15	**GOSUB** or **FOR** Line Deleted
16	RETURN Error
17	Syntax Error
18	Invalid String Character
19	LOAD program Too Long
20	Device Number Larger
21	LOAD File Error
128	BREAK Abort
129	IOCB
131	IOCB Write Only
132	Invalid Command
133	Device or File not Open
134	BAD IOCB Number
135	IOCB Read Only Error
136	EOF
137	Truncated Record
138	Device Timeout
139	Device NAK
140	Serial Bus
141	Cursor Out of Range
142	Serial Bus Data Frame Overrun
143	Serial Bus Data Frame Checksum Error
144	Device Done Error
145	Bad Screen Mode Error
146	Function Not Implemented
147	Insufficient RAM
160	Drive Number Error
161	Too many OPEN Files
162	Disk Full
163	Unrecoverable System Data I/O Error
164	File Number Mismatch
165	File Name Error
166	POINT Data Length Error
167	File Locked
168	Command Invalid
169	Directory Full
170	File Not Found
171	POINT Invalid

For explanation of Error Messages see Appendix B.

CONTENTS

PREFACE

1 **GENERAL INFORMATION**
Terminology 1
Special Notations Used In This Manual 4
Abbreviations Used In This Manual 5
Operating Modes 6
Special Function Keys 6
1200XL Keys and Indicators 7
1200XL Self Test 8
Arithmetic Operators 9
Logical Operators 9
Operator Precedence 10
Built-In Functions 10
Graphics 10
Sound and Games 10
Wraparound and Keyboard Rollover 11
Error Messages 11

2 **COMMANDS**
BYE 12
CONT 12
END 12
LET 13
LIST 13
NEW 14
REM 14
RUN 14
STOP 14

3 **EDIT FEATURES**
Screen Editing 15
Control (CTRL) Key 15
Shift Key 15
Double Key Functions 16
Cursor Control Keys 16
Keys Used With CTRL Key 16
Keys Used With Shift Key 16
Special Function Keys 16
Break Key 16
Escape Key 16
ATARI 1200XL Key Functions 17

4 **PROGRAM STATEMENTS**
FOR/NEXT/STEP 18
GOSUB/RETURN 19
GOTO 21
IF/THEN 22
ON/GOSUB 24
ON/GOTO 24
POP 25
RESTORE 27
TRAP 28

5 **INPUT/OUTPUT COMMANDS**
Input/Output Devices 29
CLOAD 30
CSAVE 30
DOS 31
ENTER 31
INPUT 31
LOAD 32
LPRINT 32
NOTE 33
OPEN/CLOSE 33
POINT 34
PRINT 34
PUT/GET 35
READ/DATA 35
SAVE 36
STATUS 36
XIO 37
Chaining Programs 38
Modifying a BASIC Program on Disk 38

6 **FUNCTION LIBRARY**
Arithmetic Functions 40
ABS 40
CLOG 40
EXP 40
INT 41
LOG 41
RND 41
SGN 41
SQR 41
Trigonometric Functions 42
ATN 42
COS 42
SIN 42
DEG/RAD 42

Special Purpose Functions 42
ADR 42
FRE 42
PEEK 43
POKE 43
USR 43

7 **STRINGS**
ASC 45
CHR$ 45
LEN 46
STR$ 46
VAL 46
String manipulations 47

8 **ARRAYS AND MATRICES**
DIM 50
CLR 51

9 **GRAPHICS MODES AND COMMANDS**
GRAPHICS 52
Graphics Modes 52
Mode 0 53
Modes 1 and 2 54
Modes 3, 5, and 7 55
Modes 4 and 6 55
Mode 8 55
Mode 9, 10 and 11 55
COLOR 56
DRAWTO 56
LOCATE 56
PLOT 57
POSITION 57
PUT/GET 57
SETCOLOR 58
XIO (Special Fill Application) 61

10 **SOUND AND GAME CONTROLLERS**
SOUND 66
PADDLE 68
PTRIG 69
STICK 69
STRIG 69

11 ADVANCED PROGRAMMING TECHNIQUES
Memory Conservation 70
Programming In Machine Language 71

APPENDIX A BASIC RESERVED WORDS 76

APPENDIX B ERROR MESSAGES 81

APPENDIX C ATASCII CHARACTER SET WITH DECIMAL/ HEXADECIMAL LOCATIONS 84

APPENDIX D ATARI 400/800/1200XL MEMORY MAP 93

APPENDIX E DERIVED FUNCTIONS 96

APPENDIX F PRINTED VERSIONS OF CONTROL CHARACTERS 97

APPENDIX G GLOSSARY 98

APPENDIX H USER PROGRAMS 102

APPENDIX I MEMORY LOCATIONS 119

APPENDIX J TABLE OF MODES AND SCREEN FORMATS 121

INDEX 123

PREFACE

This manual is not intended to "teach" BASIC. It is a reference guide to the commands, statements, functions, and special applications of ATARI® BASIC.

Many of the programs and partial programming examples used in this manual are photostats of listings printed on an ATARI printer. Some of the special symbols in the ATARI character set do not appear the same on the printer; e.g., the clear screen symbol " ↰ " appears as a " } ". The examples in the text were chosen to illustrate a particular function—not necessarily "good" programming techniques.

Each of the sections contains groups of commands, functions, or statements dealing with a particular aspect of ATARI BASIC. For instance, Section 9 contains all the statements pertaining to the unique graphics capabilities of ATARI Home Computers. The appendices include quick references to terms, error messages, BASIC keywords, memory locations, and the ATASCII character set.

As there is no one specified application for the ATARI Home Computer System, this manual is directed at general applications and the general user. Appendix H contains programs that illustrate a few of the ATARI computer system's capabilities.

This revision of the manual includes information on the ATARI 1200 XL Home Computer and the GTIA graphic modes. The ATARI 400/800 Home Computers may not contain all the features in this manual.

GENERAL INFORMATION 1

This section explains BASIC terminology, special notations, and abbreviations used in this manual, and the special keys on the ATARI Home Computer keyboard. It also points to other sections where BASIC commands deal with specific applications.

TERMINOLOGY

BASIC: **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode.

BASIC Keyword: Any reserved word "legal" in the BASIC language. May be used in a statement, as a command, or for any other purpose. (See Appendix A for a list of all "reserved words" or keywords in ATARI BASIC.)

BASIC Statement: Usually begins with a keyword, like **LET, PRINT,** or **RUN.** Keywords are shown in heavy capital letters.

Command String: Multiple commands (or program statements) placed on the same numbered line if statement numbers are used, or the same logical line if direct mode is used. The commands must be separated by colons.

Constant: A constant is a value expressed as a number rather than represented by a variable name. For example, in the statement X = 100, X is a variable and 100 is a constant. (See **Variable**.)

Expression: An expression is any legal combination of variables, constants, operators, and functions used together to compute a value. Expressions can be either arithmetic, logical, or string.

Floating Point Number: A number containing an integer part, a decimal point, and a fractional part. The total number of significant digits in a floating point number, excluding the exponent, is nine.

Function: A function is a computation built into the computer so that it can be called for by the user's program. A function is *NOT* a statement; it is part of an expression. It is really a subroutine used to compute a value which is then "returned" to the main program when the subroutine returns. **COS** (Cosine), **RND** (random), **FRE** (unused memory space), and **INT** (integer) are examples of functions. In many cases the value is simply assigned to a variable (stored in a variable) for later use. In other cases it may be printed out on the screen immediately. See Section 6 for more on functions. Examples of functions as they might appear in programs are:

```
10 PRINT RND(0)
```
(print out the random number returned)

```
10 X=100+COS(45)
```
(add the value re-returned to 100 and store the total in variable X)

Logical Line: A logical line consists of one to three physical lines, and is terminated either by the **RETURN** key or automatically when the maximum logical line limit is reached. Each numbered line in a BASIC program consists of one logical line when displayed on the screen. When entering a line which is longer than one physical line, the cursor will automatically go to the beginning of the next physical line when the end of the current physical line is reached. If **RETURN** is not entered, then both physical lines will be part of the same logical line.

Operator: Operators are used in expressions. Operators include *addition (+), subtraction (−), multiplication (*), division (/), exponentiation (∧), greater than (>), less than (<), equal to (=), greater than or equal to (> =), less than or equal to (< =), and not equal to (< >).* The logical keywords **AND, NOT** and **OR** are also operators. The + and − operators can also be used as unary operators; e.g., −3. Do *not* put more than one unary operator in a row; e.g., −−3, as the computer may interpret it incorrectly.

Physical Line: One line of characters as displayed on a television screen.

String: A string is a group of characters enclosed in quotation marks. "ABRACADABRA" is a string. So are "ATARI MAKES GREAT COMPUTERS" and "123456789". A string is much like a constant, as it too, may be stored in a variable. A string variable is different, in that its name must end in the character **$.** For example, the string "ATARI COMPUTER" may be assigned to a variable called **A$** using (optional) **LET** like this:

```
10 LET A$="ATARI COMPUTER"
20 A$="ATARI COMPUTER"
```
(note quotation marks)

(**LET** is optional; the quotes are required.)

Quotation marks may *not* be used within a string. However, the closing quotation can be omitted if it is the last character on a logical line. (See Section 7—**STRINGS**).

Variable: A variable is the name for a numerical or other quantity, which may (or may not) change. Variable names may be up to 120 characters long. However, a variable name must start with an alphabetic letter, and may contain only capital letters and numerical digits. *Do not* use a keyword as a variable name or as the first part of a variable name as it is not interpreted correctly. Examples of storing a value in a variable:

```
10 LET CI23DVB=1.234
20 LET VARIABLE112=267.543
30 LET A=1
40 LET F5TH=6.5
50 LET THISNO=59.009
```

Note: **LET** is optional and may be omitted.

Variable Name Limit: ATARI BASIC limits the user to 128 variable names. To bypass this problem, use individual elements of an array instead of having separate variable names. BASIC keeps all references to a variable that has been deleted from a program, and the name still remains in the variable name table.

If the screen displays an ERROR-4 (Too Many Variables) message, use the following procedure to make room for new variable names:

```
LIST filespec

NEW

ENTER filespec
```

The LIST filespec writes the untokenized version of the program onto a disk or cassette. NEW clears the program and the table areas. The program is then re-entered, re-tokenized, and a new variable table is built. (The tokenized version is Atari BASIC's internal format. The untokenized version is in ATASCII, which is the version displayed on the screen).

Arrays and Array Variables: An array is a list of places where data can be filed for future use. Each of these places is called an *element,* and the whole array or any element is an array variable. For example, define "Array A" as having 6 elements. These elements are referred to by the use of subscripted variables such as **A(2), A(3), A(4),** etc. A number can be stored in each element. This may be accomplished element by element (using the **LET** statement), or as a part of a **FOR/NEXT** loop (see Chapter 8).

Note: Never leave blanks between the element number in parentheses and the name of the array.

Correct	**Incorrect**
A(23)	**A (23)**
ARRAY(3)	**ARRAY (3)**
X123(38)	**X123 (38)**

SPECIAL NOTATIONS USED IN THIS MANUAL

Line Format: In deferred mode, the format of a line in a BASIC program includes a line number (abbreviated to lineno) at the beginning of the line, followed by a statement keyword, followed by the body of the statement and ending with a line terminator command (RETURN key). In an actual program, the four elements might look like this:

	STATEMENT		
Line Number	**Keyword**	**Body**	**Terminator**
100	**PRINT**	**A/X * (Z + 4.567)**	RETURN key

Several statements can be typed on the same line provided they are separated by colons (:). See **IF/THEN** in Section 4. In direct mode, the format is identical, except that no line number is used, and the statement is processed immediately after the RETURN key is pressed.

Bold Capital Letters: In this manual, denote keywords to be typed by the user in upper case form exactly as they are printed in this text. Here are a few examples:

PRINT INPUT LIST END GOTO GOSUB FOR NEXT IF

Capital Letters: In this manual, are used to identify keys on the keyboard, such as RETURN, SELECT, etc.

Lower Case Letters: In this manual, lower case letters are used to denote the various classes of items which may be used in a program, such as variables (**var**), expressions (**exp**), and the like. The abbreviations used for these classes of items are shown in Table 1-1.

Items in Brackets: Brackets, [ ], contain optional items which may be used, but are not required. If the item enclosed in brackets is followed by three dots **[,exp...]**, it means that *any* number of expressions may be entered, but none are required.

Items stacked vertically in braces: Items stacked vertically in braces indicate that any one of the stacked items may be used, but that only one at a time is permissible. In the example below, type either the **GOTO** or the **GOSUB.**

100 { **GOTO** / **GOSUB** } 2000

Command abbreviations in headings: If a command or statement has an abbreviation associated with it, the abbreviation is placed following the full name of the command in the heading; e.g., **LET (L.)**.

ABBREVIATIONS USED IN THIS MANUAL

The following table explains the abbreviations used throughout this manual:

TABLE 1-1 ABBREVIATIONS

AVAR	**Arithmetic Variable:** A location where a numeric value is stored. Variable names may be from 1 to 120 alphanumeric characters, but must start with an alphabetic character, and all characters must be unreversed and all alpha characters must be upper case.
SVAR	**String Variable:** A location where a string of characters may be stored. The same name rules as avar apply, except that the last character in the variable name must be a **$**. String variables may be subscripted. See Section 7, **STRINGS.**
MVAR	**Matrix Variable:** Also called a *Subscripted Variable.* An element of an array or matrix. The variable name for the array or matrix as a whole may be any legal variable name such as **A, X, Y, ZIP,** or **K.** The subscripted variable (name for the particular element) starts with the matrix variable, and then uses a number, variable, or expression in parentheses *immediately* following the array or matrix variable. For example, **A(ROW), A(1), A(X + 1).**
VAR	**Variable:** Any variable. May be MVAR, AVAR, or SVAR
AOP	**Arithmetic operator.** (+ − * / ^)
LOP	**Logical operator.** (NOT AND OR)
AEXP	**Arithmetic Expression:** Generally composed of a variable, function, constant, or two arithmetic expressions separated by an arithmetic operator.
LEXP	**Logical Expression:** Generally composed of two arithmetic or string expressions separated by a logical operator. Such an expression evaluates to either a 1 (logical true) or a 0 (logical false). For example, the expression 1 < 2 evaluates to the value 1 (true) while the expression "LEMON" = "ORANGE" evaluates to a zero (false) as the two strings are not equal.
SEXP	**String Expression:** Can consist of a string variable, string literal (constant), or a function that returns a string value.
EXP	Any expression, whether sexp or aexp.
LINENO	**Line Number:** A constant that identifies a particular program line in a deferred mode BASIC program. Must be any integer from 0 through 32767. Line numbering determines the order of program execution.
ADATA	**ATASCII Data:** Any ATASCII character excluding commas and carriage returns. (See Appendix C.)
FILESPEC	**File Specification:** A string expression that refers to a device such as the keyboard or to a disk file. It contains information on the type of I/O device, its number, a colon, an optional file name, and an optional filename extender. (See **OPEN,** Section 5.) **Example filespec:** "D1:NATALIE.ED"

OPERATING MODES

Direct Mode: Uses no line numbers and executes instruction immediately after **RETURN** key is pressed.

Deferred Mode: Uses line numbers and delays execution of instruction(s) until the **RUN** command is entered.

Execute Mode: Sometimes called **RUN** mode. After **RUN** command is entered, each program line is processed and executed.

Memo Pad Mode: A non-programmable mode that allows the user to experiment with the keyboard or to leave messages on the screen. Nothing written while in Memo Pad mode affects the RAM-resident program.

SPECIAL FUNCTION KEYS

Reverse (Inverse) Video key, or "ATARI LOGO KEY". This key is used on the 400/800. Pressing this key causes the text to be reversed on the screen (dark text on light background). Press key a second time to return to normal text.

Reverse (Inverse) Video key. This key is used on the 1200XL. Pressing this key causes the text to be reversed on the screen (dark text on light background). Press key a second time to return to normal text.

CAPS/LOWR — **Lower Case key:** Pressing this key on the 400/800 shifts the screen characters from upper case (capitals) to lower case. To restore the characters to upper case, press the SHIFT key and the CAPS/LOWR key simultaneously.

CAPS — **Upper/Lower Case key:** Pressing this key on the 1200XL changes the screen characters from upper to lower or the reverse each time it is pressed. The SHIFT key is *not* used.

ESC — **Escape key:** Pressing this key causes a command to be entered into a program for later execution.

Example: To clear the screen, you would enter:
10 **PRINT** "ESC CTRL CLEAR"
and press RETURN.

Escape is also used in conjunction with other keys to print special graphic control characters. See Appendix F for the specific keys and their screen-character representations.

BREAK — **Break key:** Pressing this key during program execution causes execution to stop. Execution may be resumed by typing **CONT** followed by pressing RETURN.

SYSTEM RESET — **System Reset key:** Similar to **BREAK** in that pressing this key stops program execution. Also returns the screen display to Graphics mode 0, clears the screen, and returns margins and other variables to their default values.

SET-CLR-TAB	**Tab key:** Press SHIFT and the SET-CLR-TAB keys simultaneously to set a tab. To clear a tab, press the CTRL and SET-CLR-TAB keys simultaneously. Used alone, the SET-CLR-TAB advances the cursor to the next tab position. In Deferred mode, set and clear tabs by preceding the above with a line number, the command **PRINT,** a quotation mark, and press the ESC key. **Examples:** 100 **PRINT** "ESC SHIFT SET-CLR-TAB" 200 **PRINT** "ESC CTRL SET-CLR-TAB" Default tab settings are placed at columns 7, 15, 23, 31, and 39. The leftmost screen position is column 0, but entry begins in column 2. A total of 38 columns (or character positions) can be shown in one line on the screen.
INSERT	**Insert key:** Press the SHIFT and INSERT keys simultaneously to insert a line. To insert a single character, press the CTRL and INSERT keys simultaneously.
DELETE BACK S	**Delete key:** Press the SHIFT and DELETE keys simultaneously to delete a line. To delete a single character, press CTRL and DELETE simultaneously.
DELETE BACK S	**Back Space key:** Pressing this key replaces the character to the left of the cursor with a space and moves cursor back one space.
CLEAR	**Clear key:** Pressing this key while holding down the SHIFT or CTRL key blanks the screen and puts the cursor in the upper left corner.
RETURN	**Return key:** Terminator to indicate an end of a line of BASIC. Pressing this key causes a numbered line to be interpreted and added to a BASIC program RAM. An unnumbered line (in Direct mode) is interpreted and executed immediately. Any variables are placed in a variable table.

1200XL KEYS AND INDICATORS

The keys and indicators described in this section are for the 1200XL only.

POWER-ON INDICATOR	This indicator is on when power to the computer is on.
L1 INDICATOR	The computer keyboard is disabled when this indicator is on. Refer to function key F1.
L2 INDICATOR	The computer has the European character set enabled when this indicator is on. Refer to function key F4.
FUNCTION KEY F1	This key moves the cursor up in one-line increments. It repeats if held down. If used with the shift key, the cursor moves to the upper left corner (also called "home position") of the screen. If used with the control key, it acts as a toggle to enable or disable the keyboard. LED 1 is lighted when the keyboard is disabled.

FUNCTION KEY F2	This key moves the cursor down in one-line increments. It repeats if held down. If used with the shift key, the cursor moves to the lower left corner of the screen. If used with the control key, it acts as a toggle to enable or disable the video presentation. When the video presentation is disabled, the processing speed of the 1200XL is increased.
FUNCTION KEY F3	This key moves the cursor to the left in one-space increments. It repeats if held down. If used with the shift key, the cursor moves to the left side of the screen. If used with the control key, it acts as a toggle to enable or disable the key click sound.
FUNCTION KEY F4	This key moves the cursor to the right in one-space increments. It repeats if held down. If used with the shift key, the cursor moves to the right side of the screen. If used with the control key, it allows the user to select either the domestic or European character set. Each time the 1200XL is powered up, the domestic character set is selected by the operating system. When the European character set is selected (by the user), LED 2 is lighted.
HELP KEY	This key provides user access to additional information on the operation currently in progress, if the programming for that function has been implemented.

ATARI 1200XL SELF TEST

The self-test function allows the user to verify that the 1200XL is fully operational. To begin the test, remove any cartridge and turn off any disk drive. Press **SYSTEM RESET**. A dynamic rainbow ATARI should appear on the screen. Press **HELP** to view the self-test menu.

Use the **SELECT** key to pick any or all of the tests. The selection that is flashing is the current selection. Press **START** to begin the test. The test cycles repeatedly until either the **HELP** key or the **SYSTEM RESET** key is pressed. The **HELP** key returns to the menu; the **SYSTEM RESET** key reboots the system and displays the rainbow **ATARI** again.

The memory test displays two long bars in line. Each bar represents one of the 8K ROMs that contain the operating system. If a bar turns green, the corresponding ROM is good; if the bar turns red, the ROM is bad. Immediately below the ROM test display, the RAM test is displayed in three segmented lines.

The RAM test displays a total of 48 color segments, each representing 1K of RAM. As each 1K segment is tested it is shown in white, and if it is good it turns to green. If a segment turns to red, the corresponding 1K of RAM is bad. As each segment of RAM is tested, LED1 and LED2 are turned on alternately, providing a test for them also.

The keyboard test displays a keyboard on the screen. As each key is pressed, the "key" on the screen is shown in inverse video and a tone is generated.

The audio-visual test displays a musical staff containing six notes. The test cycles through four "voices" of six notes each, generating a tone as each note is displayed.

If the **ALL TEST** is selected, the 1200 cycles through the entire range of tests continuously. The keyboard test is performed by the computer using a random selection of 10 to 20 keys being tested on the screen.

ARITHMETIC OPERATORS

The ATARI Home Computer System uses five arithmetic operators:

+ addition (also unary plus; e.g., +5)
− subtraction (also unary minus; e.g., −5)
* multiplication
/ division
^ exponentiation

LOGICAL OPERATORS

The logical operators consist of two types: *unary* and *binary*. The unary operator is **NOT.** The binary operators are:

AND Logical AND
OR Logical OR

Examples:

```
10 IF A=12 AND T=0 THEN
   PRINT "GOOD"
```

Both expressions must be true before GOOD is printed.

```
20 A=(C>1) AND (N<1)
```

If both expressions true, A = +1; otherwise A = 0.

```
30 A=(C+1) OR (N-1)
```

If either expression true, A = +1; otherwise A = 0.

```
40 A= NOT (C+1)
```

If expression is false, A = +1; otherwise A = 0.

The rest of the binary operators are relational.

< The first expression is less than the second expression.
> The first expression is greater than the second.
= The expressions are equal to each other.
< = The first expression is less than or equal to the second.
> = The first expression is greater than or equal to the second.
< > The two expressions are not equal to each other.

These operators are most frequently used in **IF/THEN** statements and logical arithmetic.

OPERATOR PRECEDENCE

Operations within the innermost set of parentheses are performed first and proceed out to the next level. When sets of parentheses are enclosed in another set, they are said to be "nested." Operations on the same nesting level are performed in the following order:

Highest precedence	<, >, =, <=, >=, <>	Relational operators used in string expressions have same precedence and are performed from left to right.
	–	Unary minus.
	^	Exponentiation.
	*, /	Multiplication and division have the same precedence level and are performed from left to right.
	+, –	Addition and subtraction have the same precedence level and are performed from left to right.
	<, >, =, <=, >=, <>	Relational operations in numeric expressions have the same precedence level from left to right.
	NOT	Unary operator
	AND	Logical AND
Lowest precedence	OR	Logical OR

BUILT-IN FUNCTIONS

The section titled **FUNCTION LIBRARY** explains the arithmetic and special functions incorporated into ATARI BASIC.

GRAPHICS

ATARI graphics include 16 graphics modes for the ATARI 1200, and 12 graphics modes for the ATARI 400 and 800 if the GTIA chip is installed, and 9 modes if the CTIA chip is installed. The commands have been designed to allow maximum flexibility in color choice and pattern variety. Section 9 explains each command and gives examples of the many ways to use each.

SOUND AND GAMES CONTROLLERS

The ATARI Home Computer is capable of emitting a large variety of sounds including simulated explosions, electronic music, and "raspberries." Section 10 defines the commands for using the SOUND function and for controlling paddle, joystick, and keyboard controllers.

WRAPAROUND, KEYBOARD ROLLOVER, AND KEY REPEAT

The ATARI Home Computer System has screen wraparound thus allowing greater flexibility. It also allows the user to type one key ahead. If the user presses and holds any key, it begins repeating after 1/2 second.

ERROR MESSAGES

If a data entry error is made, the screen display shows the line reprinted preceded by the message ERROR- and the offending character is highlighted. After correcting the character in the original line, delete the line containing the ERROR- *before* pressing **RETURN.** Appendix B contains a list of all the error messages and their definitions.

If the error line contains deferred screen edit function keys, the error message may become disoriented. Use the LIST command to edit error line.

2 COMMANDS

Whenever the cursor (□) is displayed on the screen, the computer is ready to accept input. Type the command (in either Direct or Deferred mode), and press **RETURN.** This section describes the commands used to clear computer memory and other useful control commands. The commands explained in this section are the following:

BYE **NEW**
CONT **REM**
END **RUN**
LET **STOP**
LIST

BYE (B.)

Format: **BYE**
Example: **BYE**

If you have an ATARI 400/800 Home Computer, the **BYE** command exits BASIC and puts the computer in Memo Pad mode. This allows the user to experiment with the keyboard or to leave messages on the screen without disturbing any BASIC program in memory. To return to BASIC, press **SYSTEM RESET.**

If you have an ATARI 1200XL Home Computer, the **BYE** command exits to the power-up display, the rainbow ATARI symbol. At this time you can have the 1200XL perform SELF-TEST by pressing the HELP key.

CONT (CON.)

Format: **CONT**
Example: **CONT**

Typing this command followed by a **RETURN** causes program execution to resume. If a **BREAK key is pressed, or a STOP,** or **END** command is encountered, the program stops until **CONT** command is entered. Execution resumes at the next sequential *line number* following the statement at which the program stopped.

Note: If the statement at which the program is halted has other commands on the same numbered line which were not executed at the time of the **BREAK, STOP,** or **END,** they will *not* be executed. On **CONT**, execution resumes at the next numbered line. A loop may be incorrectly executed if the program is halted before the loop completes execution.

This command has no effect in a Deferred mode program.

END

Format: **END**
Example: 1000 **END**

This command terminates program execution and is used in Deferred mode. In ATARI BASIC, an **END** is not required at the end of a program. When the end of the program is reached, ATARI BASIC automatically closes all files and turns off sounds (if any). **END** may also be used in Direct mode to close files and turn off sounds.

LET (LE.)

Format: **[LET]** var = exp
Example: **LET** X = 3.142 * 16
LET X = 2

The keyword **LET** in the example above is optional in defining variables. It can just as easily be left out of the statement. It is often used to set a variable name equal to a value.

LIST (L.)

Format: **LIST** [lineno [, lineno]]
LIST [filespec [,lineno [,lineno]]]
Examples: **LIST**
LIST 10
LIST 10,100
LIST "P",20,100
LIST "P
LIST "D:DEMO.LST"

This command causes the computer to display the source version of all lines currently in memory if the command is entered without line number(s), or to display a specified line or lines. For example, **LIST** 10,100 displays lines 10 through 100 on the screen. If the user has not typed the lines into the computer in numerical order, a **LIST** will automatically place them in order.

Typing L."P:" will print the RAM-resident program on the printer.

LIST can be used in Deferred mode as part of an error trapping routine (See **TRAP** in Section 4).

The **LIST** command is also used in recording programs on cassette tape. The second format is used and a filespec is entered. (See Section 5 for more details on peripheral devices.) If the entire program is to be listed on tape, no line numbers need be specified.

Example: **LIST** "C1"
1000 **LIST** "C1"

NEW

Format: **NEW**
Example: **NEW**

This command erases the program stored in RAM. Therefore, before typing **NEW**, either **SAVE** or **CSAVE** any programs to be recovered and used later. **NEW** clears BASIC's internal symbol table so that no arrays (See Section 8) or strings (See Section 7) are defined. Used in Direct mode.

REM (R. OR SPACE.)

Format: **REM** text
Examples: 10 **REM** ROUTINE TO CALCULATE X
10(SPACE). ROUTINE FOR DATA ("SPACE" means one press of the SPACE bar)

This command and the text following it are for the user's information only. It is ignored by the computer. However, it is included in a **LIST** along with the other numbered lines. Any statement on the same numbered line that occurs after a **REM** statement is ignored.

RUN (RU.)

Format: **RUN** [filespec]
Examples: **RUN**
RUN "D:MENU"

This command causes the computer to begin executing a program. If no filespec is specified, the current RAM-resident program begins execution. If a filespec is included, the computer retrieves the specified, tokenized program from the specified file and executes it.

All variables are set to zero and all open files and peripherals are closed. All arrays, strings, and matrices are eliminated and all sounds are turned off. Unless the **TRAP** command is used, an error message is displayed if any error is detected during execution and the program halts.

RUN can be used in Deferred mode.

Example: 10 **PRINT** "OVER AND OVER AGAIN."
20 **RUN**

Type **RUN** and press RETURN. To end, press BREAK.

To begin program execution at a point other than the first line number, type **GOTO** followed by the specific line number, then press RETURN.

STOP (STO.)

Format: **STOP**
Example: 100 **STOP**

When the **STOP** command is executed in a program, BASIC displays the message **STOPPED AT LINE** _____, terminates program execution, and returns to Direct mode. The **STOP** command does not close files or turn off sounds, so the program can be resumed by typing **CONT** and pressing the RETURN key.

EDIT FEATURES 3

In addition to the special function keys described in Section 1, there are cursor control keys that allow immediate editing capabilities. These keys are used in conjunction with the SHIFT or CTRL keys.

The following key functions are described in this section:

CTRL	CTRL INSERT	CTRL 1	CTRL F1	SHIFT F1
SHIFT	CTRL DELETE	CTRL 2	CTRL F2	SHIFT F2
CTRL	SHIFT INSERT	CTRL 3	CTRL F3	SHIFT F3
CTRL	SHIFT DELETE	BREAK	CTRL F4	SHIFT F4
CTRL	SHIFT CAPS/LOWR	ESC	F1	
CTRL			F2	
			F3	
			F4	

SCREEN EDITING

The keyboard and display are logically combined for a mode of operation known as screen editing. Each time a change is completed on the screen, the RETURN key must be pressed. Otherwise, the change is not made to the program in RAM.

Example:

```
10 REM PRESS RETURN AFTER LINE EDIT
20 PRINT :PRINT
30 PRINT "THIS IS LINE 1 ON SCREEN."
```

To delete line 20 from the program, type the line number and press the RETURN key. Merely deleting the line from the screen display does **not** delete it from the program.

The screen and keyboard as I/O devices are described in Section 5.

CTRL — **Control key.** Striking this key in conjunction with the arrow keys produces the cursor control functions that allow the user to move the cursor anywhere on the screen without changing any characters already on the screen. Other key combinations control the setting and clearing of tabs, halting and restarting program lists, and the graphics control symbols. Striking a key while holding the CTRL key will produce the upper-left symbol on those keys having three functions.

SHIFT — **Shift key:** This key is used in conjunction with the numeric keys to display the symbols shown on the upper half of those keys. It is also used in conjunction with other keys to insert and delete lines, return to a normal, upper case letter display, and to display the function symbols above the subtraction, equals, addition, and multiplication operators as well as the brackets, [ ], and question mark,?.

DOUBLE-KEY FUNCTIONS

Cursor Control Keys

CTRL ↑	Moves cursor up one physical line without changing the program or display.
CTRL →	Moves cursor one space to the right without disturbing the program or display.
CTRL ↓	Moves cursor down one physical line without changing the program or display.
CTRL ←	Moves cursor one space to the left without disturbing the program or display.

Like the other keys on the ATARI keyboard, holding the cursor control keys for more than 1/2 second causes the keys to repeat.

Keys Used With CTRL

CTRL INSERT	Inserts one character space.
CTRL DELETE	Deletes one character or space.
CTRL 1	Stops temporarily and restarts screen display without "breaking out" of the program.
CTRL 2	Rings buzzer.
CTRL 3	Indicates end-of-file.

Keys Used With SHIFT

SHIFT INSERT	Inserts one physical line.
SHIFT DELETE	Deletes one physical line.
SHIFT CAPS/LOWR	Returns screen display to upper-case alphabetic characters.

Special Function Keys

BREAK	Stops program execution or program list, prints a **STOPPED AT LINE** on the screen, and displays cursor.
ESC	Allows commands normally used in Direct mode to be placed in Deferred mode; e.g., in Direct mode, **CTRL CLEAR** clears the screen display. To clear the screen in Deferred mode, type the following after the program line number. Press **ESC** then press **CTRL** and **CLEAR** together. **PRINT** "ESC CTRL CLEAR"

ATARI 1200XL KEY FUNCTIONS

Keys Used With CTRL

CTRL F1*	Enables or disables keyboard.
CTRL F2	Enables or disables display.
CTRL F3	Enables or disables key click sound.
CTRL F4	Selects domestic or European character set.

Keys Used Alone

F1	Moves cursor up in one-line increments.
F2	Moves cursor down in one-line increments.
F3	Moves cursor to left in one-space increments.
F4	Moves cursor to right in one-space increments.

Keys Used With SHIFT

SHIFT F1	Moves cursor to upper left (home position) corner.
SHIFT F2	Moves cursor to lower left corner.
SHIFT F3	Moves cursor to left side of current line.
SHIFT F4	Moves cursor to right side of current line.

*Function Key

4 PROGRAM STATEMENTS

This section explains the commands associated with loops, conditional and unconditional branches, error traps, and subroutines and their retrieval. It also explains the means of accessing data and the optional command used for defining variables. The following commands are described in this section:

FOR, TO, STEP/NEXT	**IF/THEN**	**POP**
GOSUB/RETURN	**ON, GOSUB**	**RESTORE**
GOTO	**ON, GOTO**	**TRAP**

FOR (F.), TO, STEP/NEXT (N.)

Format: **FOR** avar = aexp1 **TO** aexp2 [**STEP** aexp3]
NEXT avar

Examples:
```
FOR X = 1 TO 10
NEXT X
FOR Y = 10 TO 20 STEP 2
NEXT Y
FOR INDEX = Z TO 100 * Z
NEXT INDEX
```

This command sets up a loop and determines how many times the loop is executed. The loop variable (avar) is initialized to the value of aexp1. Each time the **NEXT** avar statement is encountered, the loop variable is incremented by the aexp3 in the **STEP** statement. The aexp3 can be positive or negative integers, decimals, or fractional numbers. If there is *no* **STEP** aexp3 command, the loop increments by one. When the loop completes the limit as defined by aexp2, it stops and the program proceeds to the statement immediately following the **NEXT** statement; it may be on the same line or on the next sequential line.

All loops are executed at least once. Loops can be nested, one within another. In this case, the innermost loop is completed before returning to the outer loop. Figure 4-1 illustrates a nested loop program.

```
10 FOR X=1 TO 3
20 PRINT "OUTER LOOP"
30 Z=0
40 Z=Z+2
50 FOR Y=1 TO 5 STEP Z
60 PRINT "     INNER LOOP"
70 NEXT Y
80 NEXT X
90 END
```

Figure 4-1. Nested Loop Program

In Figure 4-1, the outer loop will complete three passes (X = 1 to 3). However, before this first loop reaches its **NEXT** X statement, the program gives control to the inner loop. Note that the **NEXT** statement for the inner loop must precede the **NEXT** statement for the outer loop. In the example, the inner loop's number of passes is determined by the **STEP** statement (**STEP Z**). In this case, **Z** has been defined as 0, then redefined as **Z + 2.** Using this data, the computer must complete three passes through the inner loop before returning to the outer loop. The aexp3 in the step statement could also have been defined as the numerical value 2.

The program run is illustrated in Figure 4-2.

```
READY


RUN
OUTER LOOP
     INNER LOOP
     INNER LOOP
     INNER LOOP
OUTER LOOP
     INNER LOOP
     INNER LOOP
     INNER LOOP
OUTER LOOP
     INNER LOOP
     INNER LOOP
     INNER LOOP

READY
```

Figure 4-2. Nested Loop Execution

The return address for the loops are placed in a special group of memory addresses referred to as a *stack.* The information is "**push**ed" on the stack and when used, the information is "**pop**ped" off the stack (see **POP**).

GOSUB (GOS.), RETURN (RET.)

Format: **GOSUB** lineno
lineno
RETURN

Example:
```
100 GOSUB 2000
2000 PRINT "SUBROUTINE"
2010 RETURN
```

A subroutine* is a program or routine used to compute a certain value, etc. It is generally used when an operation must be replaced several times within a program sequence using the same or different values. This command allows the user to "call" the subroutine, if necessary. The last line of the subroutine must contain a **RETURN** statement. The **RETURN** statement goes back to the next logical statement following the **GOSUB** statement.

Like the preceding **FOR/NEXT** command, the **GOSUB/RETURN** command sequence uses a stack for its return address. If the subroutine is not allowed to complete normally; e.g., a **GOTO** lineno before a **RETURN,** the **GOSUB** address must be "**pop**ped" off the stack (see **POP**) or it could cause future errors.

To prevent accidental triggering of a subroutine (which normally follows the main program), place an **END** statement preceding the subroutine. Figure 4-3 demonstrates the use of subroutines.

```
10 PRINT " "                                    (Clear screen)
20 REM EXAMPLE USE OF GOSUB/RETURN
30 X=100
40 GOSUB 1000
50 X=120
60 GOSUB 1000
70 X=50
80 GOSUB 1000
90 END
1000 Y=3*X
1010 X=X+Y
1020 PRINT X,Y
1030 RETURN
```

Figure 4-3. GOSUB/RETURN Program Listing

In the above program, the subroutine, beginning at line 1000, is called three times to compute and print out different values of X and Y. Figure 4-4 illustrates the results of executing this program.

```
RUN

 400          300
 480          360
 200          150

READY
```

Figure 4-4. GOSUB/RETURN Program Run

* Generally, a subroutine can do anything that can be done in a program. It is used to save memory and program-entering time, and to make programs easier to read and debug.

GOTO (G.)

Format: { **GO TO** } aexp
{ **GOTO** }

Examples: 100 **GOTO** 50
500 **GOTO** (X + Y)

The **GOTO** command is an unconditional branch statement just like the **GOSUB** command. They both immediately transfer program control to a target line number or arbitrary expression. However, using anything other than a constant will make renumbering the program difficult. If the target line number is non-existent, an error results. Any **GOTO** statement that branches to a preceding line may result in an "endless" loop. Statements following a **GOTO** statement will not be executed. Note that a conditional branching statement (see **IF/THEN**) can be used to break out of a **GOTO** loop. Figure 4-5 illustrates two uses of the **GOTO** command.

```
10 PRINT
20 PRINT :PRINT "ONE"
30 PRINT "TWO"
40 PRINT "THREE"
50 PRINT "FOUR"
60 PRINT "FIVE"
65 GOTO 100
70 PRINT "$$$$$$$$$$$$$$$$$$"
80 PRINT "%%%%%%%%%%%%%%%%%"
90 PRINT "??????????????????"
95 END
100 PRINT "SIX"
110 PRINT "SEVEN"
120 PRINT "EIGHT"
130 PRINT "NINE"
140 PRINT "TEN"
150 GOTO 70
```

Figure 4-5. GOTO Program Listing

Upon execution, the numbers in the above listing will be listed first followed by the three rows of symbols. The symbols listed on lines 70, 80, and 90 are ignored temporarily while the program executes the **GOTO** 100 command. It proceeds with the printing of the numbers "SIX" through "TEN", then executes the second **GOTO** statement which transfers program control back to line 70. (This is just an example. This program could be rewritten so that no **GOTO** statements were used.) The results of the program run are shown in Figure 4-6.

```
READY

RUN


ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
TEN
$$$$$$$$$$$$$$$$$$$
%%%%%%%%%%%%%%%%%%%
???????????????????
```

Figure 4-6. GOTO Program Run

IF/THEN

Format: **IF** aexp **THEN** { lineno | statement [:statement...] }

Examples: **IF** X = 100 **THEN** 150
IF A$ = "ATARI" **THEN** 200
IF AA = 145 and BB = 1 **THEN** PRINT AA, BB
IF X = 100 **THEN** X = 0

The **IF/THEN** statement is a conditional branch statement. This type of branch occurs only if certain conditions are met. These conditions may be either arithmetic or logical. If the aexp following the **IF** statement is true (non-zero), the program executes the **THEN** part of the statement. If, however, the aexp is false (a logical 0), the rest of the statement is ignored and program control passes to the next numbered line.

In the format, **IF** aexp **THEN** lineno, lineno must be a constant, not an expression and specifies the line number to go to if the expression is true. If several statements occur after the **THEN**, separated by colons, then they will be executed if *and only* if the expression is true. Several **IF** statements may be nested on the same line. For example:

```
10 IF X=5 THEN IF Y=3 THEN R=9:GOTO 100
```

The statements R = 9: **GOTO** 100 will be executed only if X = 5 and Y = 3. The statement **IF** Y = 3 will be executed if X = 5.

The program in Figures 4-7 and 4-8 demonstrates the **IF/THEN** statement.

```
5 GRAPHICS 0:? :? "       IF DEMO"
10 ? :? "ENTER A";:INPUT A
20 IF A=1 THEN 40:REM MULTIPLE STATEME
NTS HERE WILL NEVER BE EXECUTED!!
30 ? :? "A IS NOT 1. EXECUTION CONTINU
ES HERE WHEN THE EXPRESSION IS FALSE"
40 IF A=1 THEN ? :? "A=1":? "YES, IT I
S REALLY 1.":REM MULTIPLE STATEMENTS H
ERE WILL BE EXECUTED ONLY IF A=1!!
50 ? :? "EXECUTION CONTINUES HERE IF A
<>1 OR AFTER 'YES, IT IS REALLY 1' IS
DISPLAYED"
60 GOTO 10
```

Figure 4-7. IF/THEN Program

```
       IF DEMO

ENTER A?3                                    (entered 3)

A IS NOT 1. EXECUTION CONTINUES HERE W
HEN THE EXPRESSION IS FALSE

EXECUTION CONTINUES HERE IF A<>1 OR AF
TER 'YES, IT IS REALLY 1' IS DISPLAYED


ENTER A?1                                    (entered 1)

A=1
YES, IT IS REALLY 1.

EXECUTION CONTINUES HERE IF A<>1 OR AF
TER 'YES, IT IS REALLY 1' IS DISPLAYED


ENTER A?
```

Figure 4-8. IF/THEN Program Execution

ON/GOSUB/RETURN ON/GOTO

Format: **ON** aexp { **GOTO** / **GOSUB** } lineno [,lineno...]

Examples: 100 **ON** X **GOTO** 200, 300, 400
100 **ON** A **GOSUB** 1000, 2000
100 **ON SQR**(X) **GOTO** 30, 10, 100

Note: GOSUB and **GOTO** may not be abbreviated.

These two statements are also conditional branch statements like the **IF/THEN** statement. However, these two are more powerful. The aexp must evaluate to a positive number, which is then rounded to the nearest positive integer (whole number) value up to 255. If the resulting number is 1, then program control passes to the first lineno in the list following the **GOSUB** or **GOTO**. If the resulting number is 2, program control passes to the second lineno in the list, and so on. If the resulting number is 0 or is greater than the number of linenos in the list, the conditions are not met and program control passes to the next statement which may or may not be located on the same line. With **ON/GOSUB**, the selected subroutine is executed and then control passes to the next statement. The program in Figures 4-9 and 4-10 demonstrates the **ON/GOTO** statement:

```
10 X=X+1
20 ON X GOTO 100,200,300,400,500
30 IF X>5 THEN PRINT "COMPLETE":END
40 GOTO 10
50 END
100 PRINT "NOW WORKING AT LINE 100"
110 GOTO 10
200 PRINT "NOW WORKING AT LINE 200"
210 GOTO 10
300 PRINT "NOW WORKING AT LINE 300"
310 GOTO 10
400 PRINT "NOW WORKING AT LINE 400"
410 GOTO 10
500 PRINT "NOW WORKING AT LINE 500"
510 GOTO 10
```

Figure 4-9 ON/GOTO Program Listing

When the program is executed, it looks like the following:

```
RUN
NOW WORKING AT LINE 100
NOW WORKING AT LINE 200
NOW WORKING AT LINE 300
NOW WORKING AT LINE 400
NOW WORKING AT LINE 500
COMPLETE

READY
```

Figure 4-10 ON/GOTO Program Execution

POP

Format: **POP**
Example: 1000 **POP**

In the description of the **FOR/NEXT** statement, the *stack* was defined as a group of memory addresses reserved for return addresses. The top entry in the stack controls the number of loops to be executed and the **RETURN** target line for a **GOSUB**. If a subroutine is not terminated by a **RETURN** statement, the top memory location of the stack is still loaded with some numbers. If another **GOSUB** is executed, that top location needs to be cleared. To prepare the stack for a new **GOSUB**, use a **POP** to clear the data from the top location in the stack.

The **POP** command must be used according to the following rules:

1. It must be in the execution path of the program.
2. It must follow the execution of any **GOSUB** statement that is not brought back to the main program by a **RETURN** statement.

The program in Figure 4-11 demonstrates the use of the **POP** command with a **GOSUB** when the **RETURN** is not executed:

```
10 REM THIS PROGRAM DEMONSTRATES THE
20 REM USE OF 'POP' WITH A 'GOSUB'
30 REM WHEN A 'RETURN' IS NOT USED
40 REM TO EXIT A SUBROUTINE.
50 GOSUB 200
60 PRINT "PROPER USE OF POP INSURES"
70 PRINT "THE PROGRAM WILL RETURN"
80 PRINT "TO THIS MESSAGE."
90 END
170 REM **************
180 REM 1ST SUBROUTINE
190 REM **************
200 PRINT "EXECUTING 1ST SUBROUTINE."
210 PRINT
```

```
220 GOSUB 800
230 REM *** 'POP' NOT USED ***
240 PRINT "NOT USING POP WILL CAUSE"
250 PRINT "AN INCORRECT RETURN TO"
260 PRINT "THIS MESSAGE."
270 PRINT "THIS IS A RESULT OF"
280 PRINT "LEAVING AN EXTRA RETURN"
290 PRINT "ADDRESS ON THE STACK."
300 STOP
490 REM *** EXIT WITHOUT 'RETURN' ***
500 PRINT "THIS MESSAGE IS PRINTED"
510 PRINT "FROM THE EXITING THE 2ND"
520 PRINT "SUBROUTINE WITHOUT USING"
530 PRINT "THE 'RETURN' STATEMENT."
540 PRINT
550 REM REMOVE LINE 570 TO SEE THE
560 REM RESULT OF NOT USING 'POP'.
570 POP
580 RETURN
770 REM **************
780 REM 2ND SUBROUTINE
790 REM **************
800 PRINT "EXECUTING 2ND SUBROUTINE."
810 PRINT
820 GOTO 500
```

Figure 4-11. GOSUB Statement With POP

RESTORE (RES.)

Format: **RESTORE** [aexp]
Example: 100 **RESTORE**

The ATARI Home Computer System contains an internal "pointer" that keeps track of the **DATA** statement item to be read next. Used without the optional aexp, the **RESTORE** statement resets that pointer to the first data item in the program. Used with the optional aexp, the **RESTORE** statement sets the pointer to the first data item on the line specified by the value of the aexp. This statement permits repetitive use of the same data. (Figure 4-12).

```
10 FOR N=1 TO 2
20 READ A
30 RESTORE
40 READ B
50 M=A+B
60 PRINT "TOTAL EQUALS";M
70 NEXT N
80 END
90 DATA 30,15
```

Figure 4-12. Restore Program Listing

On the first pass through the loop, **A** will be 30 and **B** will be 30 so the total line 60 will print **TOTAL EQUALS 60,** but on the second pass, **A** will equal 15 and **B,** because of the **RESTORE** statement, will still equal 30. Therefore, the **PRINT** statement in line 60 will display **TOTAL EQUALS 45.**

The **RESTORE** statement will not generate an error if the line number referenced does not exist. Instead it will **RESTORE** to the next larger line number in the program. Care should be taken to update **RESTORE** statements when renumbering a BASIC program.

TRAP (T.)

Format: **TRAP** aexp
Example: 100 **TRAP** 120

The **TRAP** statement is used to direct the program to a specified line number if an error is detected. Without a **TRAP** statement, the program stops executing when an error is encountered and displays an error message on the screen.

The **TRAP** statement works on any error that may occur after it has been executed, but once an error has been detected and trapped, it is necessary to reset the trap with another **TRAP** command. This **TRAP** command may be placed at the beginning of the section of code that handles input from the keyboard so that the **TRAP** is reset after each error. **PEEK**(195) will give you an error message (see Appendix B). 256*__PEEK__(187) + __PEEK__(186) will give you the number of the line where the error occurred. The **TRAP** may be cleared by executing a **TRAP** statement with an aexp whose value is from 32767 to 65535 (e.g., 40000).

INPUT/OUTPUT COMMANDS AND DEVICES 5

This section describes the input/output devices and how data is moved between them. The commands explained in this section are those that allow access to the input/output devices. The input commands are those associated with getting data into the RAM and the devices geared for accepting input. The output commands are those associated with retrieving data from RAM and the devices geared for generating output.

The commands described in this section are:

CLOAD	**INPUT**	**OPEN/CLOSE**	**READ/DATA**
CSAVE	**LOAD**	**POINT**	**SAVE**
DOS	**LPRINT**	**PRINT**	**STATUS**
ENTER	**NOTE**	**PUT/GET**	**XIO**

INPUT/OUTPUT DEVICES

The hardware configuration of each of the following devices is illustrated in the individual manuals furnished with each. The Central Input/Output (CIO) subsystem provides the user with a single interface to access all of the system peripheral devices in a (largely) independent manner. This means there is a single entry point and a device-independent calling sequence. Each device has a symbolic device name used to identify it; e.g., **K:** for the keyboard. Each device must be opened before access and each must be assigned to an Input/Output Control Block (IOCB). From then on, the device is referred to by its IOCB number.

ATARI BASIC contains 8 blocks in RAM which identifies to the Operating System the information it needs to perform an I/O operation. This information includes the command, buffer length, buffer address, and two auxiliary control variables. ATARI BASIC sets up the IOCBs, but the user must specify which IOCB to use. BASIC reserves IOCB #0 for I/O to the Screen Editor, therefore the user may not request IOCB #0. The **GRAPHICS** statement (see Section 9) opens IOCB #6 for input and output to the screen. (This is the graphics window **S:**). IOCB #7 is used by BASIC for the **LPRINT**, **CLOAD**, and **CSAVE** commands. The IOCB number may also be referred to as the device (or file) number. IOCBs 1 through 5 are used in opening the other devices for input/output operations. If IOCB #7 is in use, it prevents **LPRINT** or some of the other BASIC I/O statements from being performed.

Keyboard: (K:) Input only device. The keyboard allows the user to read the converted (ATASCII) keyboard data as each key is pressed.

Line Printer: (P:) Output only device. The line printer prints ATASCII characters, a line at a time. It recognizes no control characters.

Program Recorder: (C:) Input and Output device. The recorder is a read/write device which can be used as either, but never as both simultaneously. The cassette has two tracks for sound and program recording purposes. The audio track cannot be recorded from the ATARI system, but may be played back through the television speaker.

Disk Drives: (D1:, D2:, D3:, D4:) Input and Output devices. If 16K of RAM is installed, the ATARI can use from one to four disk drives. If only one disk drive is attached, there is no need to add a number after the symbolic device code D. If D: is used, with no drive number specified, the ATARI system defaults to drive 1.

Screen Editor: (E:) Input and Output device. This device uses the keyboard and display (see *TV Monitor*) to simulate a screen editing terminal. Writing to this device causes data to appear on the display starting at the current cursor position. Reading from this device activates the screen editing process and allows the user to enter and edit data. Whenever the **RETURN** key is pressed, the entire logical line within which the cursor resides is selected as the current record to be transferred by CIO to the user program. (See Section 9).

TV Monitor: (S:) Input and Output device. This device allows the user to read characters from and write characters to the display, using the cursor as the screen addressing mechanism. Both text and graphics operations are supported. See Section 9 for a complete description of the graphics modes.

Interface, RS-232: (R:) The RS-232 device enables the ATARI system to interface with RS-232-compatible devices such as printers, terminals, and plotters. It contains a parallel port to which the 80-column printer can be attached. If a printer is attached to the parallel port, the R: is not required, and P: can be used as it is with other printers.

CLOAD (CLOA.)

Format: **CLOAD**
Examples: **CLOAD**
100 **CLOAD**

This command can be used in either Direct or Deferred mode to load a program from cassette tape into RAM for execution. On entering **CLOAD**, a buzzer sounds to indicate that the PLAY button needs to be pressed followed by the **RETURN** key. However, do not press PLAY until after the tape has been positioned. Specific instructions for **CLOAD**ing a program are contained in the *ATARI Program Recorder Manual.* Steps for loading oversized programs are included in the paragraphs under **CHAINING PROGRAMS** at the end of this section.

CSAVE (CS.)

Format: **CSAVE**
Examples: **CSAVE**
100 **CSAVE**
100 **CS.**

This command is usually used in Direct mode to save a RAM-resident program onto cassette tape. **CSAVE** saves the tokenized version of the program. On entering **CSAVE** two buzzers sound to indicate that the PLAY and RECORD buttons must be pressed followed by the **RETURN** key. Do not, however, press the buttons until the tape has been positioned. It is faster to save a program using this command rather than a **SAVE "C"** (see **SAVE**) because short inter-record gaps are used.

Notes: Tapes saved using the two commands, **SAVE** and **CSAVE**, are not compatible.

It may be necessary to enter an **LPRINT** (see **LPRINT**) before using **CSAVE**. Otherwise, **CSAVE** may not work properly.

For specific instructions on how to connect and operate the hardware, cue the tape, etc., see the *ATARI Program Recorder Manual.*

DOS (DO.)

Format: **DOS**
Example: **DOS**

The **DOS** command is used to go from BASIC to the Disk Operating System (**DOS**). If the Disk Operating System has not been booted into memory, the computer will go into Memo Pad mode (or power-on display in 1200XL) and the user must press SYSTEM RESET to return to Direct mode. If the Disk Operating System has been booted, the **DOS** Menu is displayed. To clear the **DOS** Menu from the screen, press SYSTEM RESET. Control then passes to BASIC. Control can also be returned to BASIC by selecting **B** (Run Cartridge) on the **DOS** Menu.

The **DOS** command is usually used in Direct mode; however, it may be used in a program. For more details on this, see the *ATARI **DOS** Manual.*

ENTER (E.)

Format: **ENTER** filespec
Examples: **ENTER** "C
ENTER "D:DEMOPR.INS"

This statement causes a cassette tape to play back a program originally recorded using **LIST** (see Section 2, **LIST**). The program is entered in unprocessed (untokenized) form, and is interpreted as the data is received. When the loading is complete, it may be run in the normal way. The **ENTER** command may also be used with the disk drive. Note that both **LOAD** and **CLOAD** (see Section 2) clear the old program from memory before loading the new one. **ENTER** merges the old and new programs. The **ENTER** statement is usually used in Direct mode.

INPUT (I.)

Format: **INPUT** [#aexp {,;}] {avar svar} [, {avar svar} ...]
Examples: 100 **INPUT** X
100 **INPUT** N$
100 **PRINT** "ENTER THE VALUE OF X": **INPUT** X
110 **INPUT** X

This statement requests keyboard data from the user. In execution, the computer displays a ? prompt when the program encounters an **INPUT** statement. It is usually preceded by a **PRINT** statement that prompts the user as to the type of information being requested.

String variables are allowed only if they are not subscripted. Matrix variables are not allowed.

The #aexp is optional and is used to specify the file or device number from which the data is to be input (see Input/Output Devices). If no #aexp is specified, then input is from the screen editor (**E:**).

If several strings are to be input from the screen editor, type one string, press **RETURN**, type the next string, **RETURN**, etc. Arithmetic numbers can be typed on the same line separated by commas. A typical input program is shown in Figure 5-1.

```
10 ? "ENTER 5 NUMBERS TO BE SUMMED"
20 FOR N=1 TO 5
30 INPUT X
40 C=C+X
50 NEXT N
60 ? "THE SUM OF THE NUMBERS IS    ";C
70 END
```

Figure 5-1 Input Program Listing

When executing an **INPUT** from the screen, avoid moving the cursor away from and then back to the same line; otherwise, the wrong data may be input.

If a string of 128–255 characters is **INPUT**, then RAM locations 1536–1664 will be overwritten. This area is normally reserved for storage of programs or data. To **INPUT** strings of more than 127 characters, use the **GET** command and store the values into a string (see Section 5, **OPEN/CLOSE** and **PUT/GET** commands).

Note: The maximum number of characters that can be **INPUT** from the screen is 120. The maximum for other devices is 255.

LOAD (LO.)

Format: **LOAD** filespec
Example: **LOAD** "D1:JANINE.BRY"

This command is similar to **CLOAD** except the full file name system can be used. **LOAD** uses long inter-record gaps on the tape (see **CLOAD**) and uses the tokenized version of the program. When using only one disk drive, it is not necessary to specify a number after the "D" because the default is disk drive #1.

LPRINT (LP.)

Format: **LPRINT** [exp] [{ ; / , } exp...]
Example: **LPRINT** "PROGRAM TO CALCULATE X"
100 **LPRINT** X;" ";Y;" ";Z

This statement causes the computer to print data on the line printer rather than on the screen. It can be used in either Direct or Deferred modes. It requires no device specifier and no **OPEN** or **CLOSE** statement. (BASIC uses IOCB #7.) To print a program listing on the line printer, see **LIST.**

Note: An **LPRINT** command with a semicolon at the end causes various results depending on the printer in use. To use the semicolon effectively, use the **OPEN** statement for the printer, then write to the printer with a **PRINT** statement (see **OPEN/CLOSE** and **PRINT** commands, Section 5).

NOTE (NO.)

Format: **NOTE** #aexp, avar, avar
Example: 100 **NOTE** #1, X, Y

This command is used to store the current disk sector number in the first avar and the current byte number within the sector in the second avar. This is the current read or write position in the specified file where the next byte to be read or written is located. This **NOTE** command is used when writing data to a disk file (see **POINT**). The information in the **NOTE** command can be written into a second file which is then used as an index into the first file.

OPEN (O.), CLOSE (CL.)

Formats: **OPEN** #aexp,aexp1,aexp2, filespec
CLOSE #aexp
Examples: 100 **OPEN** #2,8,0,"D1:ATARI.BAS"
100 A$ = "D1:ATARI.BAS"
110 **OPEN** #2,8,0,A$
150 **CLOSE** #2

Before a device can be accessed, it must be opened. This "**open**ing" process links a specific IOCB to the appropriate device handler, initializes any CIO-related control variables, and passes any device-specific options to the device handler. The parameters for the **OPEN** command are defined as follows:

#
Mandatory character that must be entered by the user.

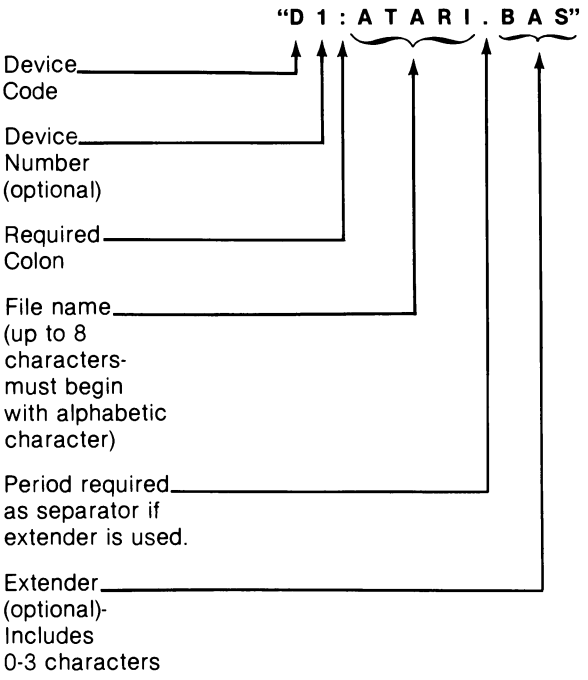
aexp
Reference IOCB or file number to same parameters for future use (as in **CLOSE** command). Number may be 1 through 7.

aexp1
Code number to determine input or output operation.

Code 4 = input operation
8 = output operation
12 = input and output operation
6 = disk directory input operation (In this case, the filespec is the search specification.)
9 = end-of-file append (output) operation. Append is also used for a special screen editor input mode. This mode allows a program to input the next line from E: without waiting for the user to press RETURN.

aexp2
Device-dependent auxiliary code. An 83 in this parameter indicates sideways printing on a printer (see appropriate manuals for control codes).

filespec
Specific file designation. Must be enclosed in quotation marks. The format for the filespec parameter is shown in Figure 5-2.


"D 1 : A T A R I . B A S"
Device Code
Device Number (optional)
Required Colon
File name (up to 8 characters- must begin with alphabetic character)
Period required as separator if extender is used.
Extender (optional)- Includes 0-3 characters

Note: Filenames are not used with the program recorder.

Figure 5-2 Filename Breakdown

Note: Be sure to include the closing quotation marks on a filespec parameter, especially when putting multiple statements on one line. For example,

OPEN #1, 4, 0, **"D:TEST":STOP** will work, but
OPEN #1, 4, 0, **"D:TEST:STOP** will not function correctly.

The **CLOSE** command simply closes files that have been previously opened with an **OPEN** command. Note in the example that the aexp following the mandatory # character must be the same as the aexp reference number in the **OPEN** statement.

POINT (P.)

Format: **POINT** #aexp, avar, avar
Example: 100 **POINT** #2, A,B

This command is used when reading a file into RAM or writing a file from RAM. The first avar specifies the sector number and the second avar specifies the byte within that sector where the next byte will be read or written. Essentially, it moves a software-controlled pointer to the specified location in the file. This gives the user "random" access to the data stored on a disk file. The **POINT** and **NOTE** commands are discussed in more detail in the ***DOS** Manual.*

Note: To update a file, you must open it with a 12 in aexp1.

PRINT (PR. or ?)

Format: **PRINT** [#aexp] [{ ; , } exp...]
Examples: **PRINT** X, Y, Z, A$
100 **PRINT** "THE VALUE OF X IS ";X
100 **PRINT** "COMMAS", "CAUSE", "COLUMN", "SPACING"
100 **PRINT** #3, A$
100 **PRINT** 2 + 3 + 4

A **PRINT** command can be used in either Direct or Deferred mode. In Direct mode, this command prints whatever information is contained between the quotation marks exactly as it appears. In the first example, **PRINT** X,Y,Z,A$, the screen will display the current values of X,Y,Z, and A$ as they appear in the RAM-resident program. In the example, **PRINT** #3,A$, the #3 is the file specifier (may be any number between 1 and 7) that controls to which device the value of A$ will be printed. (See **Input/Output Devices.**)

A comma tabs every ten spaces. Several commas in a row cause several tab jumps. A semicolon causes the next aexp or sexp to be placed immediately after the preceding expression with no spacing. Therefore, in the second example a space is placed before the ending quotation mark so the value of X will not be placed immediately after the word "IS". If no comma or semicolon is used at the end of a **PRINT** statement, then a RETURN is output and the next **PRINT** starts on the following line.

However, if the last character to be printed (as in a string with quotation marks) is a CTRL **R** or CTRL **U,** then the next **PRINT** begins at the end of the current line.

The **PRINT** command can be used as a one-line calculator in Direct mode, as shown in the last example above. In this case the value is computed when the RETURN key is pressed, and the value is printed on the next line.

Note: In rare circumstances data printed to a diskette may have part of the BASIC program embedded in it. If this occurs, retry the operation.

PUT (PU.)/GET (GE.)

Format: **PUT** #aexp, aexp
GET #aexp, aexp
Examples: 100 **PUT** #6, ASC("A")
200 **GET** #1,X

The **PUT** and **GET** are opposites. The **PUT** command outputs a single byte from 0-255 to the file specified by #aexp. (# is a mandatory character in both these commands). The **GET** command reads one byte from 0-255 (using #aexp to designate the file, etc. on diskette or elsewhere) and then stores the byte in the variable avar.

READ (REA.), DATA (D.)

Format: **READ** var [, var...]
DATA adata [, adata...]
Examples: 100 **READ** A,B,C,D,E
110 **DATA** 12,13,14,15,16
100 **READ** A$,B$,C$,D$,E$
110 **DATA** EMBEE, EVELYN, CARLA, CORINNE, BARBARA

These two commands are always used together and the **DATA** statement is always used in Deferred mode[1]. The **DATA** statement can be located anywhere in the program, but must contain as many pieces of data as there are defined in the **READ** statement. Otherwise, an "out of data" error is displayed on the screen. Refer to **RESTORE** command.

String variables used in **READ** statements must be dimensioned and cannot be subscripted. (See **STRINGS** Section). Neither may array variables be used in a **READ** statement.

The **DATA** statement holds a number of string data for access by the **READ** statement. It cannot include arithmetical operations, functions, etc. Furthermore, the data type in the **DATA** statement must match the variable type defined in the corresponding **READ** statement. The program in Figure 5-3 totals a list of numbers in a **DATA** statement:

```
10 FOR N=1 TO 5
20 READ D
30 M=M+D
40 NEXT N
50 PRINT "SUM TOTAL EQUALS          ";M
60 END
70 DATA 30,15,106,17,87
```

Figure 5-3 Read/Data Program Listing

The program, when executed, will print the statement:

SUM TOTAL EQUALS 255.

SAVE (S.)

Format: **SAVE** filespec
Example: **SAVE** "D1:YVONNE.PAT"

The **SAVE** command is similar to the **CSAVE** command except that the full file name system can be used. The device code number is optional when using disk drive #1, because the default is to disk drive #1. **SAVE,** like **LOAD**, uses long inter-record gaps on the cassette (see **CSAVE**) and the tokenized form of the program.

STATUS (ST.)

Format: **STATUS** #aexp,avar
Example: 350 **STATUS** #1,Z

The **STATUS** command calls the **STATUS** routine for the specified device (aexp). The routine checks the device for an error condition and stores the appropriate status data in the specified variable (avar). Refer to Appendix B. An error code of 1 is stored if the device is in a ready state and no error condition is detected.

[1]A Direct mode READ will only read data if a DATA statement was executed in the program.

XIO (X.)

Format: **XIO** cmdno, #aexp, aexp1, aexp2, filespec
Example: **XIO** 18,#6,12,0,"S:"

The **XIO** command is a general input/output statement used for special operations. One example is its use to fill an area on the screen between plotted points and lines with a color (see Section 9). When a STATUS REQUEST operation is done on an OPEN device, the aexp1 used in the STATUS REQUEST must be the same as the IOCB number used in the OPEN statement for that device; e.g., if the OPEN was OPEN #1,9,0,"D:TEMP.BAS" then the STATUS REQUEST must be XIO 13,#1,9,0,"D:TEMP.BAS". The parameters for the XIO command are defined as follows:

(cmdno = Number that stands for the particular command to be performed.)

XIO cmdno	OPERATION	EXAMPLE	COMMENTS
3	OPEN	XIO 3,#1,4,0,"D:TEMP.BAS"	Same as BASIC OPEN
12	CLOSE	XIO 12,#1,0,0,"D:"	Same as BASIC CLOSE
13	STATUS REQUEST	XIO 13,#1,4,0,"D:TEMP.BAS"	See note below
17	DRAW LINE	XIO 17,#6,12,0,"S:"	See Section 9
18	FILL	XIO 18,#6,12,0,"S:"	See Section 9
32	RENAME FILE	XIO 32,#1,0,0,"D:TEMP,CAROL"	See note below
33	DELETE FILE	XIO 33,#1,0,0,"D:TEMP.BAS"	
35	LOCK FILE	XIO 35,#1,0,0,"D:TEMP.BAS"	
36	UNLOCK FILE	XIO 36,#1,0,0,"D:TEMP.BAS"	
254	FORMAT	XIO 254,#1,0,0,"D:"	

aexp — Device number (same as in OPEN). Most of the time it is ignored, but must be preceded by #.

aexp1
aexp2 — Two auxiliary control bytes. Their usage depends on the particular device and command. In most cases, they are unused and are set to 0. Aexp1 should be set to 12 for a **DRAW LINE** or a **FILL** operation to allow color checking later in the program.

filespec — String expression that specifies the device. Must be enclosed in quotation marks. Although some commands, like **FILL** (Section 9), do not look at the filespec, it must still be included in the statement. XIO commands 5, 7, 9, and 11, 37, and 38, should not be used, because they are undefined and unpredictable errors might occur.

NOTE: When using the RENAME operation, the device code D: should only be used once.

D:TEMP, CAROL is correct
D:TEMP,D:CAROL is incorrect

Status Request performs the same action as the BASIC STATUS but does not return the error code in a variable. If an error condition is detected, it stops the program and prints an error message. To prevent the stopping of the program use a **TRAP** before using XIO 13. The only advantage XIO 13 has over STATUS is that a specific file on a disk drive can be checked by XIO 13 but not by STATUS.

CHAINING PROGRAMS

If a program requires more memory than is available, use the following steps to string programs of less than the maximum memory available into one program.

1. Type in the first part of the program in the normal way.
2. The last line of the first part of the program should contain only the line number and the command **RUN**"C:"
3. Cue the tape to the blank section. Write down the program counter number for later **RUN** purposes. Press PLAY and RECORD buttons on the deck so that both remain down.
4. Type **SAVE**"C:" and press the RETURN key.
5. When the beeping sound occurs, press RETURN again.
6. When the screen displays "READY", do not move tape. Type **NEW** and press RETURN.
7. Repeat the above instructions for the second part of the program.
8. As the second part of the program is essentially a totally new program, it is possible to re-use the line numbers used in the first part of the program.
9. If there is a third part of the program, make sure the last line of the second part is a **RUN**"C:" command.

To execute a "chained" program, use the following steps:

1. Cue the tape to the beginning of part 1 of the program.
2. Press PLAY button on the recorder.
3. Type **RUN**"C:"RETURN.
4. When the "beep" sounds, press RETURN again.

The computer automatically loads the first part of the program, runs it, and sounds a "beep" to indicate when to hit the space bar or RETURN to trigger the tape motor for the second **LOAD/RUN**. The loading takes a few seconds.

Note: A one-part program can be recorded and reloaded in the same way or **CSAVE** and **CLOAD** can be used.

Note: Remember to boot **DOS** *before* typing in your program if you wish to store the program on diskette.

MODIFYING A BASIC PROGRAM ON DISK

The procedure for modifying an existing BASIC program stored on a diskette is demonstrated in the following steps:

1. Turn off ATARI console and insert BASIC cartridge.
2. Connect disk drive and turn it on—without inserting diskette.
3. Wait for Busy Light to go out and for the drive to stop. Open disk drive door.
4. Insert diskette (with **DOS**) and close door.
5. Turn on console. **DOS** should boot in and the screen show READY.
6. To load program from disk, type
   LOAD "D:filename.ext
7. Modify program (or type in new program).
8. To save program on disk, type
   SAVE "D:filename.ext
9. Always wait for the Busy light to go out before removing diskette.

10. To get a Directory listing, leave the diskette in and type
    DOS
    Press **RETURN**, and the **DOS** Menu is displayed. Select command letter **A,** type it, and press **RETURN** twice to list the directory on the screen; or type **A** followed by pressing **RETURN** then type **P:** and press **RETURN** to list directory on the printer.
11. To return to BASIC, type **B** and press **RETURN** or press **SYSTEM RESET.**

6 FUNCTION LIBRARY

This section describes the arithmetic, trigonometric, and special purpose functions incorporated into the ATARI BASIC. A function performs a computation and returns the result (usually a number) for either a print-out or additional computational use. Included in the trigonometric functions are two statements, radians (RAD) and degrees (DEG), that are frequently used with trigonometric functions. Each function described in this section may be used in either Direct or Deferred mode. Multiple functions are perfectly legal.

The following functions and statements are described in this section:

ABS	**ATN**	**ADR**
CLOG	**COS**	**FRE**
EXP	**SIN**	**PEEK**
INT	**DEG/RAD**	**POKE**
LOG		**USR**
RND		
SGN		
SQR		

ARITHMETIC FUNCTIONS

ABS

Format: **ABS** (aexp)
Example: 100 AB = **ABS** (– 190)

Returns the absolute value of a number without regard to whether it is positive or negative. The returned value is always positive.

CLOG

Format: **CLOG** (aexp)
Example: 100 C = **CLOG**(83)

Returns the logarithm to the base 10 of the variable or expression in parentheses. **CLOG**(0) gives an error and **CLOG**(1) equals 0.

EXP

Format: **EXP** (aexp)
Example: 100 **PRINT** EXP(3)

Returns the value of e (approximately 2.71828283), raised to the power specified by the expression in parentheses. In the example given above, the number returned is 20.0855365. In some cases, EXP is accurate only to six significant digits.

INT

Format: **INT** (aexp)
Examples: 100 I = **INT**(3.445) *(3 would be stored in I)*
100 X = **INT**(– 14.66778) *(– 15 would be stored in X)*

Returns the greatest integer less than or equal to the value of the expression. This is true whether the expression evaluates to a positive or negative number. Thus, in our first example above, I is used to store the number 3. In the second example, X is used to store the number – 15 (the first whole number that is less than or equal to – 14.66778). This **INT** function should not be confused with the function used on calculators that simply truncates (cuts off) all decimal places.

LOG

Format: **LOG**(aexp)
Example: 100 L = **LOG**(67.89/2.57)

Returns the natural logarithm of the number or expression in parentheses. **LOG**(0) gives an error and **LOG**(1) equals 0.

RND

Format: **RND**(aexp)
Example: 10 A = **RND** (0)

Returns a hardware-generated random number between 0 and 1, but never returns 1. The variable or expression in parentheses following **RND** is a dummy and has no effect on the numbers returned. However, the dummy variable must be used. Generally, the **RND** function is used in combination with other BASIC statements or functions to return a number for games, decision making, and the like. The following is a simple routine that returns a random number between 0 and 999.

```
10 X = RND(0)
20 RX = INT(1000*X)
30 PRINT RX
```

(0 is the dummy variable)

SGN

Format: **SGN**(aexp)
Example: 100 X = **SGN**(– 199) (– 1 would be returned)

Returns a – 1 if aexp evaluates to a negative number; a 0 if aexp evaluates to 0, or a 1 if aexp evaluates to a positive number.

SQR

Format: **SQR**(aexp)
Example: 100 **PRINT SQR**(100) (10 would be printed)

Returns the square root of the aexp which must be positive.

TRIGONOMETRIC FUNCTIONS

ATN

Format: **ATN**(aexp)
Example: 100 X = **ATN**(65)

Returns the arctangent of the variable or expression in parentheses.

COS

Format: **COS**(aexp)
Example: 100 C = **COS**(X + Y + Z)

Note: Presumes X, Y, Z previously defined!

Returns the trigonometric cosine of the expression in parentheses.

SIN

Format: **SIN**(aexp)
Example: 100 X = **SIN**(Y)

Note: Presumes Y previously defined.

Returns the trigonometric sine of the expression in parentheses.

DEG/RAD

Format: **DEG**
RAD
Example: 100 **DEG**
100 **RAD**

These two statements allow the programmer to specify degrees or radians for trigonometric function computations. The computer defaults to radians unless **DEG** is specified. Once the **DEG** statement has been executed, **RAD** must be used to return to radians.

See Appendix E for the additional trigonometric functions that can be derived.

SPECIAL PURPOSE FUNCTIONS

ADR

Format: **ADR**(svar)
Example: **ADR**(A$)

Returns the decimal memory address of the string specified by the expression in parentheses. Knowing the address enables the programmer to pass the information to **USR** routines, etc. (See **USR** and Appendix D)

FRE

Format: **FRE**(aexp)
Examples: **PRINT FRE** (0)
100 **IF FRE** (0) < 1000 **THEN PRINT** "MEMORY CRITICAL"

This function returns the number of bytes of user RAM left. Its primary use is in Direct mode with a dummy variable (0) to inform the programmer how much memory space remains for completion of a program. Of course **FRE** can also be used within a BASIC program in Deferred mode.

PEEK

Format: **PEEK**(aexp)
Examples: 1000 **IF PEEK**(4000) = 255 **THEN PRINT** "255"
100 **PRINT** "LEFT MARGIN IS"; **PEEK** (82)

Returns the contents of a specified memory address location (aexp). The address specified must be an integer or an arithmetic expression that evaluates to an integer between 0 and 65535 and represents the memory address in decimal notation (not hexadecimal). The number returned will also be a decimal integer with a range from 0 to 255. This function allows the user to examine either RAM or ROM locations. In the first example above, the **PEEK** is used to determine whether location 4000 (decimal) contains the number 255. In the second example, the **PEEK** function is used to examine the left margin.

POKE

Format: **POKE** aexp1, aexp2
Examples: **POKE** 82, 10
100 **POKE** 82, 20

Although this is not a function, it is included in this section because it is closely associated with the **PEEK** function. The **POKE** command inserts data into the memory location or modifies data already stored there. In the above format, aexp1 is the decimal address of the location to be poked and aexp2 is the data to be poked. Note that this number is a decimal number between 0 and 255. **POKE** cannot be used to alter ROM locations. In gaining familiarity with this command it is advisable to look at the memory location with a **PEEK** and write down the contents of the location. Then, if the **POKE** doesn't work as anticipated, the original contents can be poked into the location.

The above Direct mode example changes the left screen margin from its default position of 2 to a new position of 10. In other words, the new margin will be 8 spaces to the right. To restore the margin to its normal default position, press SYSTEM RESET.

USR

Format: **USR** (aexp1 [, aexp2] [, aexp3...])
Example: 100 **RESULT** = **USR** (ADD1,A*2)

This function returns the results of a machine-language subroutine. The first expression, aexp1, must be an integer or arithmetic expression that evaluates to an integer that represents the decimal memory address of the machine language routine to be performed. The input arguments aexp2, aexp3, etc., are optional. These should be arithmetic expressions within a decimal range of 0 through 65535. A non-integer value may be used; however, it will be rounded to the nearest integer.

These values will be converted from BASIC's Binary Coded Decimal (BCD) floating point number format to a two-byte binary number, then pushed onto the hardware stack, composed of a group of RAM memory locations under direct control of the 6502 microprocessor chip. Figure 6-1 illustrates the structure of the hardware stack.

N (Number of arguments on the stack-may be 0)
$\mathbf{X_1}$ (High byte of argument X)
$\mathbf{X_2}$ (Low byte of argument X)
$\mathbf{Y_1}$ (High byte of argument Y)
$\mathbf{Y_2}$ (Low byte of argument Y)
$\mathbf{Z_1}$ (High byte of argument Z)
$\mathbf{Z_2}$ (Low byte of argument Z)
•
•
•
$\mathbf{R_1}$ (Low byte of return address)
$\mathbf{R_2}$ (High byte of return address)

Figure 6-1. Hardware Stack Definition

Note: X is the argument following the address of the routine, Y is the second, Z is the third, etc. There are N pairs of bytes.

See Section 11 for a description of the **USR** function in machine language programming. Appendix D defines the bytes in RAM available for machine language programming.

STRINGS 7

This section describes strings and the functions associated with string handling. Each string must be dimensioned (see **DIM** statement, Section 8) and each string variable must end with a **$**. A string itself is a group of characters "strung" together. The individual characters may be letters, numbers, or symbols (including the ATARI special keyboard symbols.) A substring is a part of a longer string and any substring is accessible in ATARI BASIC if the string has been properly dimensioned (see end of section). The characters in a string are indexed from 1 to the current string length, which is less than or equal to the dimensioned length of the string.

The string functions described in this section are:

ASC STR$ CHR$ VAL LEN

ASC

Format: **ASC**(sexp)
Examples: 100 A = **ASC**(A$)

This function returns the ATASCII code number for the first character of the string expression (sexp). This function can be used in either Direct or Deferred mode. Figure 7-1 is a short program illustrating the **ASC** function.

```
10 DIM A$(3)
20 A$="E"
30 A=ASC(A$)
40 PRINT A
```

Figure 7-1. ASC Function Program

When executed, this program prints a 69 which is the ATASCII code for the letter "E". Note that when the string itself is used, it must be enclosed in quotation marks.

CHR$

Format: **CHR$** (aexp)
Examples: 100 **PRINT CHR$** (65)
100 A$ = **CHR$** (65)

This character string function returns the character, in string format, represented by the ATASCII code number in parentheses. Only one character is returned. In the above examples, the letter A is returned. Using the **ASC** and **CHR$** functions, the program in Figure 7-2 prints the upper case letters of the alphabet.

```
10 FOR I=0 TO 25
20 PRINT CHR$(ASC("A")+I)
30 NEXT I
```

Figure 7-2. ASC and CHR$ Program Example

Note: There can be only one **STR$** and only one **CHR$** in a logical comparison. For example, A = CHR$(1) < CHR$(2) is not a valid operation.

LEN

Format: **LEN** (sexp)
Example: 100 **PRINT LEN**(A$)

This function returns the length in bytes of the designated string. This information may then be printed or used later in a program. The length of a string variable is simply the index for the character which is currently at the end of the string. Strings have a length of 0 until characters have been stored in them. It is possible to store into the middle of the string by using subscripting. However, the beginning of the string will contain garbage unless something was stored there previously.

The routine in Figure 7-3 illustrates one use of the **LEN** function:

```
10 DIM A$(10)
20 A$="ATARI"
30 PRINT LEN(A$)
```

Figure 7-3. LEN Function Example

The result of running the above program would be 5.

STR$

Format: **STR$** (aexp)
Example: A$ = **STR$**(65)

This string form number function returns the string form of the number in parentheses. The above example would return the actual number 65, but it would be recognized by the computer as a string.

Note: There can only be one **STR$** in a logical comparison. For example, A = **STR$**(1) > **STR$**(2) is not valid and will not work correctly.

VAL

Format: **VAL**(sexp)
Example: 100 A = **VAL**(A$)

This function returns a number of the same value as the number stored as a string. This is the opposite of a **STR$** function. Using this function, the computer can perform arithmetic operations on strings as shown in the example program in Figure 7-4.

```
10 DIM B$(5)
20 B$="10000"
30 B=SQR(VAL(B$))
40 ? "THE SQUARE ROOT OF ";B$;"IS";B
```

Figure 7-4. VAL Function Program

Upon execution, the screen displays THE SQUARE ROOT OF 10000 IS 100.

It is not possible to use the **VAL** function with a string that does not start with a number, or that cannot be interpreted by the computer as a number. It can, however, interpret floating point numbers; e.g., **VAL**("1E9") would return the number 1000000000.

Only the numeric field will be translated, while the text will be ignored. For example:

A$ = "5 SUM"
VAL(A$) = 5

STRING MANIPULATIONS

Strings can be manipulated in a variety of ways. They can be split, concatenated, rearranged, and sorted. The following paragraphs describe the different manipulations.

STRING CONCATENATION

Concatenation means putting two or more strings together to form one large string. Each string to be included in a larger string is called a *substring.* Each substring must be dimensioned (see **DIM**). In ATARI BASIC, a substring can contain up to 99 characters (including spaces). After concatenation, the substrings can be stored in another string variable, printed, or used in later sections of the program. Figure 7-5 is a sample program demonstrating string concatenation. In this program, A$, B$, and C$ are concatenated and placed in A$.

```
10 DIM A$(100),B$(100),C$(100)
20 A$="STRINGS/SUBSTRINGS ARE DISCUSSED"
30 B$="IN ATARI BASIC--A SELF TEACHING GUIDE"
40 C$="---CHAPTER 9"
50 A$(LEN(A$)+1)=B$
60 A$(LEN(A$)+1)=C$
70 PRINT A$
```

Figure 7-5. String Concatenation Example

STRING SPLITTING

The format of a subscript string variable is as follows:

svar(aexp1[,aexp2])

The svar is used to indicate the unsubscripted string variable name (with $). aexp1 indicates the starting location of the substring and aexp2 (if used) indicates the ending location of the substring. If no aexp2 is specified, then the end of the substring is the current end of the string. The starting location cannot be greater than the current length of the string. The two example programs in Figure 7-6 illustrate a split string with no end location indicated and a split string with an ending location indicated.

```
10 DIM S$(5)
20 S$="ABCD#"
30 PRINT S$(2)
40 END
```

Result is BCD#.
(without ending location)

```
10 DIM S$(20)
20 S$="ATARI 800 BASIC"
30 PRINT S$(7,8)
40 END
```

Result is 80
(with ending location)

Figure 7-6. Split String Examples

STRING COMPARISONS AND SORTS

In string comparisons, the logical operators are used exactly the way they are with numbers. The second program in Appendix H is a simple example of bubble sort.

In using logical operators, remember that each letter, number, and symbol is assigned an ATASCII code number. A few general rules apply to these codes:

1. ATASCII codes for numbers are sized in order of the numbers' real values and are always lower than the codes for letters (see Appendix C).
2. Upper case letters have lower numerical values than the lower case letters. To obtain the ATASCII code for a lower case letter if you know the upper case value, add 32 to the upper case code.

Note: ATARI BASIC's memory management system moves strings around in memory to make room for new statements. This causes the string address to vary if a program is modified or Direct mode is used.

ARRAYS AND MATRICES 8

An array is a one-dimensional list of numbers assigned to subscripted variables; e.g., A(0), A(1), A(2). Subscripts range from 0 to the dimensioned value. Figure 8-1 illustrates a 7-element array.

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)
A(6)

Figure 8-1. Example of an Array

A matrix, in this context, is a two-dimensional table containing rows and columns. Rows run horizontally and columns run vertically. Matrix elements are stored by BASIC in row-major order. This means that all the elements of the first row are stored first, followed by all the elements of the second row, etc. Figure 8-2 illustrates a 7 x 4 matrix.

	Columns			
Rows	M(0,0)	M(0,1)	M(0,2)	M(0,3)
	M(1,0)	M(1,1)	M(1,2)	M(1,3)
	M(2,0)	M(2,1)	M(2,2)	M(2,3)
	M(3,0)	M(3,1)	M(3,2)	M(3,3)
	M(4,0)	M(4,1)	M(4,2)	M(4,3)
	M(5,0)	M(5,1)	M(5,2)	M(5,3)
	M(6,0)	M(6,1)	M(6,2)	M(6,3)

Figure 8-2. Example of a Matrix

This section describes the two commands associated with arrays, matrices, and strings, and how to load both arrays and matrices. The commands in this section are:

DIM
CLR

DIM (DI.)

Format: **DIM** { svar(aexp) / mvar(aexp[,aexp]) } [{ ,svar(aexp) / ,mvar(aexp[,aexp]) } ...]

Examples: **DIM** A(100)
DIM M(6,3)
DIM B$(20) used with STRINGS

A **DIM** statement is used to reserve a certain number of locations in memory for a string, array, or matrix. A character in a string takes one byte in memory and a number in an array takes six bytes. The first example reserves 101 locations for an array designated A. The second example reserves 7 rows by 4 columns for a two-dimensional array (matrix) designated M. The third example reserves 20 bytes designated B$. **All strings, arrays, and matrices must be dimensioned.** It is a good habit to put all **DIM** statements at the beginning of the program. Notice in Figure 8-1 that although the array is dimensioned as **DIM** A(6), there are actually 7 elements in the array because of the 0 element. Although Figure 8-2 is dimensioned as **DIM** M(6,3), 28 locations are reserved.

Note: The ATARI Home Computer does not automatically initialize array or matrix variables to 0 at the start of program execution. To initialize array or matrix elements to 0, use the following program steps:

```
250 DIM A(100)
300 FOR E=0 TO 100
310 A(E)=0
320 NEXT E
```

Arrays and matrices are "filled" with data by using **FOR/NEXT** statements, **READ/DATA** statements and **INPUT** commands. Figure 8-3 illustrates the "building" of part of an array using the **FOR/NEXT** loop and Figure 8-4 builds an array using the **READ/DATA** statements.

```
10 DIM A(100)
20 X=10
30 FOR E=1 TO 90
40 X=X+1
50 A(E)=X
60 NEXT E
70 FOR E=1 TO 90
80 PRINT E,A(E)
90 NEXT E
```

Figure 8-3. Use of FOR/NEXT to Build An Array

```
10 DIM A(3)
20 FOR E=1 TO 3
30 READ X
40 A(E)=X
50 PRINT A(E),
60 NEXT E
70 END
100 DATA 33,45,12
```

Figure 8-4. Use of READ/DATA to Build An Array

Figure 8-5 shows an example of building a 6 x 3 matrix.

```
10 DIM M(5,2)
20 FOR ROW=0 TO 5
30 FOR COL=0 TO 2
40 M(ROW,COL)=INT(RND(0)*1000)
50 NEXT COL:NEXT ROW
60 FOR ROW=0 TO 5
70 FOR COL=0 TO 2
80 PRINT M(ROW,COL)
90 NEXT COL:PRINT :NEXT ROW
```

Figure 8-5. Building A Matrix

Note that the words ROW and COLUMN are not BASIC commands, statements, functions, or keywords. They are simply variable names used here to designate which loop function is first. The program could just as easily have been written with X and Y as the variable names.

Note: The command **COM** is identical to **DIM** and may be used in its place.

Note: Due to a discrepancy in boundary checking, arrays of up to 32766 by 32766 in size can be dimensioned. The programmer should size the array ahead of time to ensure that there is enough RAM storage space.

CLR

Format: **CLR**
Example: 200 **CLR**

This command clears the memory of all previously dimensioned strings, arrays, and matrices so the memory and variable names can be used for other purposes. If a matrix, string, or array is needed after a **CLR** command, it must be redimensioned with a **DIM** command.

9 GRAPHICS MODES AND COMMANDS

This section describes the ATARI BASIC commands and the different graphics modes of the ATARI Home Computer. Using these commands, it is possible to create graphics for graphic displays or games.

The commands to be described in this section are:

GRAPHICS	**LOCATE**	**PUT/GET**
COLOR	**PLOT**	**SETCOLOR**
DRAWTO	**POSITION**	**XIO**

The **PUT/GET** and **XIO** commands explained in this section are special applications of the same commands described in Section 5.

GRAPHICS (GR.)

Format: **GRAPHICS** aexp

Examples:
GRAPHICS 2
100 **GRAPHICS** 5 + 16
170 **GRAPHICS** 1 + 32 + 16
120 **GRAPHICS** 8
150 **GRAPHICS** 0
140 **GRAPHICS** 18

This command is used to select one of the graphics modes. The 1200XL provides 16 graphics modes; the 400/800 provide 12 graphics modes if the GTIA chip is installed and 9 if the CTIA chip is installed. Table 9-1 summarizes the modes and the characteristics of each. The **GRAPHICS** command automatically opens the screen, S:(the graphics window), as device #6. So when printing text in the text window, it is not necessary to specify the device code. The aexp must be positive, rounded to the nearest integer. Graphics mode 0 is a full-screen display while modes 1 through 8 are split screen displays. To override the split-screen, add the characters + 16 to the mode number (aexp) in the **GRAPHICS** command. Adding 32 prevents the **GRAPHICS** command from clearing the screen.

To return to graphics mode 0 in Direct mode, press SYSTEM RESET or type **GR.0** and press RETURN.

TABLE 9-1 TABLE OF MODES AND SCREEN FORMATS

SCREEN FORMAT							
Graphics Mode	Mode Type	Columns	Rows—Split Screen**	Rows—Full Screen	Number of Colors	RAM Required (Bytes) Split	Full
0	TEXT	40	—	24	1-1/2		992
1	TEXT	20	20	24	5	674	672
2	TEXT	20	10	12	5	424	420
3	GRAPHICS	40	20	24	4	434	432
4	GRAPHICS	80	40	48	2	694	696
5	GRAPHICS	80	40	48	4	1174	1176
6	GRAPHICS	160	80	96	2	2174	2184
7	GRAPHICS	160	80	96	4	4190	4200
8	GRAPHICS	320	160	192	1-1/2	8112	8138
9*	GRAPHICS	80	—	192	1		8138
10*	GRAPHICS	80	—	192	9		8138
11*	GRAPHICS	80	—	192	16		8138
12***	GRAPHICS	40	20	24	5	1154	1152
13***	GRAPHICS	40	10	12	5	664	660
14***	GRAPHICS	160	160	192	2	4270	4296
15***	GRAPHICS	160	160	192	4	8112	8138

*GTIA Mode Only
**Refer to Figure 9-1.
***1200XL Only

GRAPHICS MODE 0

This mode is the 1-color, 2-luminance (brightness) default mode for the ATARI Home Computer. It contains a 24 by 40 character screen matrix. The default margin settings at 2 and 39 allow 38 characters per line. Margins may be changed by poking LMARGN and RMARGN (82 and 83). See Appendix I. Some systems have different margin default settings. The color of the characters is determined by the background color. Only the luminance of the characters can be different. This full-screen display has a blue display area bordered in black (unless the border is specified to be another color). To display characters at a specified location, use one of the following two methods.

Method 1.

lineno **POSITION** aexp1, aexp2	*Puts cursor at location specified by aexp1 and* aexp2.
lineno **PRINT** sexp	

Method 2

lineno **GR**. 0	*Specifies graphics mode.*
lineno **POKE** 752,1	*Suppresses cursor.*
lineno **COLOR** ASC(sexp)	*Specifies character to be printed.*
lineno **PLOT** aexp1,aexp2	*Specifies where to print character.*
lineno **GOTO** lineno	*Start loop to prevent READY from being printed. (***GOTO** *same lineno.)*
	Press **BREAK** *to terminate loop.*

GRAPHICS 0 is also used as a clear screen command either in Direct mode or Deferred mode. It terminates any previously selected graphics mode and returns the screen to the default mode (**GRAPHICS 0**).

GRAPHICS MODES 1 AND 2

As defined in Table 9-1, these two 5-color modes are Text modes. However, they are both split-screen (see Figure 9-1) modes. Characters printed in Graphics mode 1 are twice the width of those printed in Graphics 0, but are the same height. Characters printed in Graphics mode 2 are twice the width and height of those in Graphics mode 0. In the split-screen mode, a **PRINT** command is used to display characters in either the text window or the graphics window. To print characters in the graphics window, specify device #6 after the **PRINT** command.

Example:
```
100 GR. 1
110 PRINT#6;"ATARI"
```

The default colors depend on the type of character input. Table 9-2 defines the default color and color register used for each type.

TABLE 9-2 DEFAULT COLORS FOR SPECIFIC INPUT TYPES

Character Type	Color Register	Default Color
Upper case alphabetical	0	Orange
Lower case alphabetical	1	Light Green
Inverse upper case alphabetical	2	Dark Blue
Inverse lower case alphabetical	3	Red
Numbers and delimiters	0	Orange
Inverse numbers and delimiters	2	Dark Blue

Note: See **SETCOLOR** to change character colors.

Unless otherwise specified, all characters are displayed in upper case non-inverse form. To print lower case letters and graphics characters, use a **POKE 756,226.** To return to upper case, use **POKE 756,224.**

In Graphics modes 1 and 2, there is no inverse video, but it is possible to get all of the characters in four different colors (see end of section).

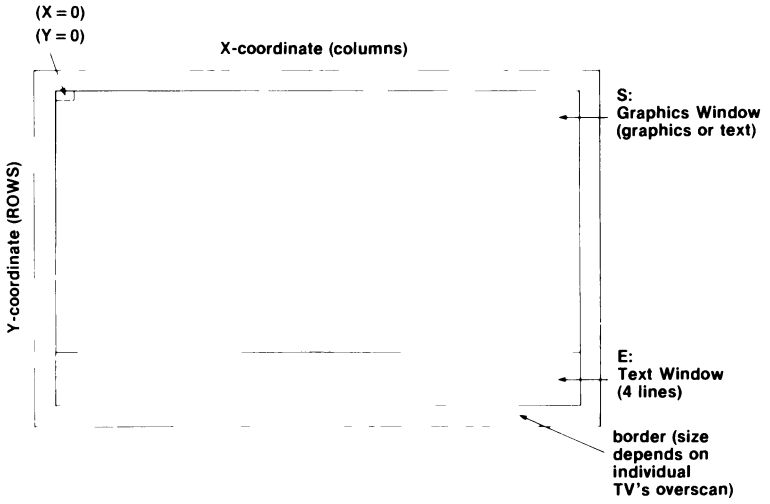


Figure 9-1. Split-Screen Display For Graphics Modes 1 and 2

As shown in Figure 9-1, the X and Y coordinates start at 0 (upper left of screen). The maximum values are the numbers of rows and columns minus 1 (see Table 9-1).

This split-screen configuration can be changed to a full screen display by adding the characters + 16 to the mode number.

Example: **GRAPHICS** 1 + 16

GRAPHICS MODES 3, 5, AND 7

These three 4-color Graphics modes are also split-screen displays in their default state, but may be changed to full screen by adding + 16 to the mode number. Modes 3, 5, and 7 are alike except that modes 5 and 7 use more points (pixels) in plotting, drawing, and positioning the cursor; the points are smaller, thereby giving a much higher resolution.

GRAPHICS MODES 4 AND 6

These two 2-color Graphics modes are split-screen displays and can display in only two colors while the other modes can display 4 and 5 colors. The advantage of a two-color mode is that it requires less RAM space (see Table 9-1). Therefore, it is used when only two colors are needed and RAM is getting crowded. These two modes also have a higher resolution which means smaller points than Graphics mode 3.

GRAPHICS MODE 8

This Graphics mode gives the highest resolution of all the modes. As it takes a lot of RAM to obtain this kind of resolution, it can only accommodate a maximum of one color and two different luminances.

GRAPHICS MODES 9, 10, AND 11

Use **GRAPHICS** to select one of the Graphics modes (9 through 11). **GRAPHICS** 9 through 11 are only available if your system has a GTIA chip. **GRAPHICS** 9 allows you to have one playfield color with 16 luminances. **GRAPHICS** 10 can have nine playfield colors with eight luminances. **GRAPHICS** 11 can have 16 colors with one luminance.

COLOR (C.)

Format: **COLOR** aexp
Examples: 110 **COLOR** ASC("A")
110 **COLOR** 3

The value of the expression in the **COLOR** statement determines the data to be stored in the display memory for all subsequent **PLOT** and **DRAWTO** commands until the next **COLOR** statement is executed. The value must be positive and is usually an integer from 0 through 4. Modes 9 through 11 use 4 bits, so the color statement varies between 0 and 15. The actual color displayed depends on the value in the color register, which corresponds to the data of 0, 1, 2, or 3 in the particular graphics mode being used. This may be determined by looking in Table 9-5, which gives the default colors and the corresponding register numbers. Colors may be changed by using **SETCOLOR**.

Note that when BASIC is first powered up, the color data is 0, and when a **GRAPHICS** command (without + 32) is executed, all of the pixels are set to 0. Therefore, nothing seems to happen to **PLOT** and **DRAWTO** in **GRAPHICS** 3 through 7 when no **COLOR** statement has been executed. Correct by doing a **COLOR** 1 first.

DRAWTO (DR.)

Format: **DRAWTO** aexp1, aexp2
Example: 100 **DRAWTO** 10,8

This statement causes a line to be drawn from the last point displayed by a **PLOT** (see **PLOT**) to the location specified by aexp1 and aexp2. The first expression represents the X coordinate and the second represents the Y-coordinate (see Figure 9-1). The color of the line is determined by the color command in effect at the time.

LOCATE (LOC.)

Format: **LOCATE** aexp1, aexp2, var
Example: 150 **LOCATE** 12, 15, X

This command positions the invisible graphics cursor at the specified location in the graphics window, retrieves the color data at that pixel, and stores it in the specified arithmetic variable. This gives a number from 0 to 255 for Graphics modes 0 through 2; 0 or 1 for the 2-color graphics modes; and 0, 1, 2, or 3 for the 4-color modes. The two arithmetic expressions specify the X and Y coordinates of the point. **LOCATE** is equivalent to:

POSITION aexp1, aexp2:**GET** #6,avar

Doing a **PRINT** after a **LOCATE** or **GET** from the screen may cause the data in the pixel which was examined to be modified. This problem is avoided by repositioning the cursor and putting the data that was read, back into the pixel before doing the **PRINT**. The program in Figure 9-2 illustrates the use of the **LOCATE** command.

```
10 GRAPHICS 3+16
20 COLOR 1
30 SETCOLOR 2,10,8
40 PLOT 10,15
50 DRAWTO 15,15
60 LOCATE 12,15,X
70 PRINT X
```

Figure 9-2. Example Program Using LOCATE

On execution, the program prints the data (**1**) determined by the **COLOR** statement which was stored in pixel 12, 15.

PLOT (PL.)

Format: **PLOT** aexp1, aexp2
Example: 100 **PLOT** 5,5

The **PLOT** command is used in Graphics modes 3 through 11 to display a point in the graphics window. The aexp1 specifies the X-coordinate and the aexp2 the Y-coordinate. The color of the plotted point is determined by the last **COLOR** statement executed. To change the color and luminance of the plotted point, use **SETCOLOR**. Points that can be plotted on the screen are dependent on the Graphics mode being used. The range of points begins at 0 and extends to one less than the total number of rows (X-coordinate) or columns (Y-coordinate) shown in Table 9-1.

POSITION (POS.)

Format: **POSITION** aexp1, aexp2
Example: 100 **POSITION** 8, 12

The **POSITION** statement is used to place the cursor (invisible in graphics mode) at a specified location on the screen. This statement usually precedes a **PRINT** statement and can be used in all modes. Note that the cursor does not actually move until an I/O command which involves the screen is issued.

PUT/GET (PU./GE.)

Formats: **PUT** #aexp, aexp
GET #aexp, avar
Examples: 100 **PUT** #6, ASC("A")
200 **GET** #1, X

In graphics work, **PUT** is used to output data to the screen display. This statement works hand-in-hand with the **POSITION** statement. After a **PUT** (or **GET**), the cursor is moved to the next location on the screen. Doing a **PUT** to device #6 causes the one-byte input (second aexp) to be displayed at the cursor position. The byte is either an ATASCII code byte for a particular character (modes 0–2) or the color data (modes 3–11).

GET is used to input the code byte of the character displayed at the cursor position, into the specified arithmetic variable. (**PRINT** and **INPUT** may also be used.)

Note: Doing a **PRINT** after a **LOCATE** or **GET** from the screen may cause the data in the pixel which was examined to be modified. To avoid this problem, reposition the cursor and put the data that was read, back into the pixel before doing the **PRINT**.

SETCOLOR (SE.)

Format: **SETCOLOR** aexp1, aexp2, aexp3
Example: 100 **SETCOLOR** 0, 1, 4

This statement is used to choose the particular hue and luminance to be stored in the specified color register. The parameters of the **SETCOLOR** statement are defined below:

aexp1 = Color register (0–4 depending on graphics mode)
aexp2 = Color hue number (0–15. See Table 9-3)
aexp3 = Color luminance (must be an even number between 0 and 14; the higher the number the brighter the display. 14 is almost pure white.)

TABLE 9-3 THE ATARI HUE (SETCOLOR COMMAND) NUMBERS AND COLORS

COLORS	SETCOLOR (aexp2) NUMBERS
GRAY	0
LIGHT ORANGE (GOLD)	1
ORANGE	2
RED-ORANGE	3
PINK	4
PURPLE	5
PURPLE-BLUE	6
BLUE	7
BLUE	8
LIGHT BLUE	9
TURQUOISE	10
GREEN-BLUE	11
GREEN	12
YELLOW-GREEN	13
ORANGE-GREEN	14
LIGHT ORANGE	15

Note: Colors vary with type and adjustment of TV or monitor used.

The ATARI display hardware contains five color registers, numbered from 0 through 4. The Operating System (OS) has five RAM locations (COLOR 0 through COLOR 4, see Appendix I—Memory Locations) where it keeps track of the current colors. The **SETCOLOR** statement is used to change the values in these RAM locations. (The OS transfers these values to the hardware registers every television frame.) The **SETCOLOR** statement requires a value from 0 to 4 to specify a color register. The **COLOR** statement uses different numbers because it specifies data which only *indirectly* corresponds to a color register. This can be confusing, so careful experimentation and study of the various tables in this section is advised.

No **SETCOLOR** commands are needed if the default set of colors is used. The purpose of the color registers and **SETCOLOR** statement is to specify the colors.

TABLE 9-4 TABLE OF SETCOLOR "DEFAULT" COLORS*

Setcolor (Color Register)	Defaults To Color	Luminance	Actual Color
0	2	8	ORANGE
1	12	10	GREEN
2	9	4	DARK BLUE
3	4	6	PINK OR RED
4	0	0	BLACK

*"DEFAULT" occurs if no **SETCOLOR** statement is used.

Note: Colors may vary depending upon the television monitor type, condition, and adjustment.

A program illustrating Graphics mode 3 and the commands explained so far in this section is shown below:

```
10 GRAPHICS 3
20 SETCOLOR 0,2,8:COLOR 1
30 PLOT 17,1:DRAWTO 17,10:DRAWTO 9,18
40 PLOT 19,1:DRAWTO 19,18
50 PLOT 20,1:DRAWTO 20,18
60 PLOT 22,1:DRAWTO 22,10
70 DRAWTO 30,18
80 POKE 752,1
90 ? :? "          ATARI HOME COMPUTERS"
100 GOTO 100
```

The **SETCOLOR** and **COLOR** statements set the color of the points to be plotted (see Table 9-5). The **SETCOLOR** command loads color register 0 with hue 2 (orange) and a luminance of 8 ("normal"). The next 4 lines plot the points to be displayed. Line 80 prints the string expression ATARI HOME COMPUTERS in the text window.

Note that the background color was never set because the default is the desired color (black).

If the program is executed, it prints the ATARI logo in the graphics window and the string expression in the text window as in Figure 9-3.

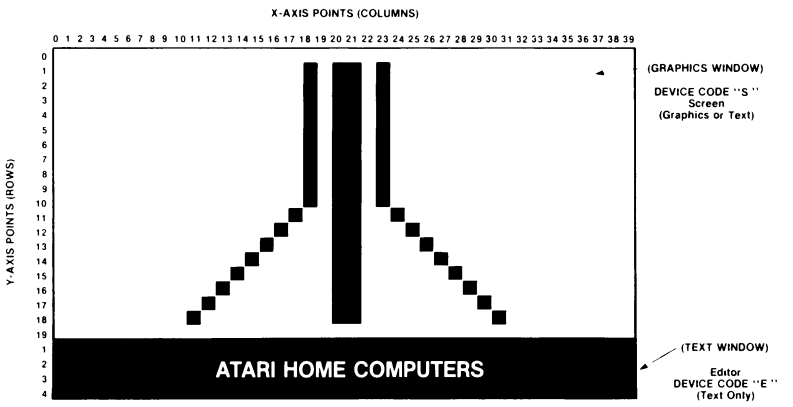

X-AXIS POINTS (COLUMNS)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
Y-AXIS POINTS (ROWS)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
(GRAPHICS WINDOW)
DEVICE CODE "S"
Screen
(Graphics or Text)
ATARI HOME COMPUTERS
(TEXT WINDOW)
Editor
DEVICE CODE "E"
(Text Only)

Figure 9-3. Atari Logo Program Execution

TABLE 9-5 MODE, SETCOLOR, COLOR TABLE

Default Colors	Mode or Condition	SETCOLOR (aexp1) Color Register No.	COLOR (aexp)	DESCRIPTION AND COMMENTS
		0	COLOR data	—
LIGHT BLUE	MODE 0 and	1	actually	Character luminance
	ALL TEXT WINDOWS		determines character to	(same color as background)
	(1 Color		be plotted	
DARK BLUE	2 Luminances)	2		Background
		3		—
BLACK		4		Border
ORANGE		0	COLOR data	Character
LIGHT GREEN	MODES 1	1	actually deter-	Character
DARK BLUE	and	2	mines character	Character
RED	2	3	to be plotted	Character
BLACK	(Text Modes)	4		Background, Border
ORANGE		0	1	Graphics point
LIGHT GREEN	MODES 3, 5,	1	2	Graphics point
DARK BLUE	and 7	2	3	Graphics point
	(Four-color	3	—	—
BLACK	Modes)	4	0	Graphics point (background default), Border
ORANGE	MODES 4	0	1	Graphics point
	and 6	1	—	—
	(Two-color	2	—	—
	Modes)	3	—	—
BLACK		4	0	Graphics point (background default), Border
		0	—	—
		1	1	Graphics point luminance (same color as background)
LIGHT BLUE	MODE 8	2	0	Graphics point (background default)
DARK BLUE	(1 Color	3	—	—
BLACK	2 Luminances)	4	—	Border
BLACK	MODE 9 (1 Color 16 Luminances)	4	0–15	Graphics point— Color value determines luminance
BLACK	MODE 10	—	0	Background (Color 0)
BLACK	(9 Color)	—	1	Graphics point
BLACK		—	2	Graphics point
BLACK		—	3	Graphics point
ORANGE		0	4	Graphics point
LIGHT GREEN		1	5	Graphics point
DARK BLUE		2	6	Graphics point
RED		3	7	Graphics point
BLACK		4	8	Graphics point
BLACK	MODE 11 (16 Colors 1 Luminance)	4	0–15	Graphics point— Color value determines hue

XIO (X.) SPECIAL FILL APPLICATION

Format: **XIO** 18, #aexp, aexp1, aexp2, filespec
Example: 100 **XIO** 18, #6, 12, 0, "S:"

This special application of the **XIO** statement uses XIO 18 fills an area on the screen between plotted points and lines with a non-zero color. A dummy variable (0) is used for aexp2. Refer to **XIO** statement for further information.

The following steps illustrate the fill process:

1. **PLOT** bottom right corner (point 1).
2. **DRAWTO** upper right corner (point 2). This outlines the right edge of the area to be filled.
3. **DRAWTO** upper left corner (point 3).
4. **POSITION** cursor at lower left corner (point 4).
5. **POKE** address 765 with the fill color data (1, 2, or 3).
6. This method is used to fill each horizontal line from top to bottom of the specified area. The fill starts at the left and proceeds across the line to the right until it reaches a pixel which contains non-zero data (will wraparound if necessary). This means that fill cannot be used to change an area which has been filled in with a non-zero value, as the fill will stop. The fill command will go into an infinite loop if a fill with zero (0) data is attempted on a line which has no non-zero pixels. **BREAK** or **SYSTEM RESET** can be used to stop the fill if this happens.

The program in Figure 9-4 creates a shape and fills it with a data (color) of 3. Note that the XIO command draws in the lines at the left and bottom of the figure.

```
10 GRAPHICS 5+16
20 COLOR 3
30 PLOT 70,45
40 DRAWTO 50,10
50 DRAWTO 30,10
60 POSITION 10,45
70 POKE 765,3
80 XIO 18,#6,12,0,"S:"
90 GOTO 90
```

Figure 9-4. Example "FILL" Program

XIO (X.) DRAW LINE APPLICATION

Format: **XIO** 17, #aexp, aexp1, aexp2, filespec
Example: 130 **XIO** 17, #6, 12, 0, "S:"

This application of the XIO statement uses XIO 17 and draws a line on the screen between the last point plotted and the current position of the (invisible) graphics cursor (moved by the POSITION command) in the current color.

```
100 GRAPHICS 5: COLOR 2
110 PLOT 5,5
120 POSITION 10,10
130 XIO 17, #6, 12, 0,"S:"
```

The above program draws a line from 5,5 to 10,10 in COLOR 2. Lines 120 and 130 could be replaced by

```
120 DRAWTO 10,10
```

TABLE 9-6 INTERNAL CHARACTER SET

Column 1				Column 2			
#	CHR	#	CHR	#	CHR	#	CHR
0	Space	16	0	32	@	48	P
1	!	17	1	33	A	49	Q
2	"	18	2	34	B	50	R
3	#	19	3	35	C	51	S
4	$	20	4	36	D	52	T
5	%	21	5	37	E	53	U
6	&	22	6	38	F	54	V
7	'	23	7	39	G	55	W
8	(	24	8	40	H	56	X
9	)	25	9	41	I	57	Y
10	*	26		42	J	58	Z
11	+	27	;	43	K	59	[
12	,	28	<	44	L	60	\
13	—	29	=	45	M	61	]
14		30	>	46	N	62	^
15	/	31	?	47	O	63	—

Assigning Colors To Characters In Text Modes 1 and 2

This procedure describes the method of assigning colors to the ATARI character set. First, look up the character number in Table 9-6. Then, see Table 9-7 to get the conversion of that number required to assign a color register to it.

Example: Assign **SETCOLOR** 0 to lower case "r" in mode 2 whose color is determined by register 0.

1. In Table 9-6, find the column and number for "r" (114-column 4).

Column 3				Column 4			
#	CHR	#	CHR	#	CHR	#	CHR
64		80		96		112	p
65		81		97	a	113	q
66		82		98	b	114	r
67		83		99	c	115	s
68		84		100	d	116	t
69		85		101	e	117	u
70		86		102	f	118	v
71		87		103	g	119	w
72		88		104	h	120	x
73		89		105	i	121	y
74		90		106	j	122	z
75		91 ①		107	k	123	
76		92 ①		108	l	124	|
77		93 ①		109	m	125 ①	
78		94 ①		110	n	126 ①	
79		95 ①		111	o	127 ①	

1. In mode 0 these characters must be preceded with an escape, CHR$(27), to be printed.

TABLE 9-7 CHARACTER/COLOR ASSIGNMENT

		Column 1 Conversion	Column 2 Conversion	Column 3 Conversion	Column 4 Conversion
MODE 0	[2]SETCOLOR 2	# + 32	# + 32	# – 64	NONE
		POKE 756,224		POKE 756,226	
MODE 1	SETCOLOR 0	# + 32	# + 32	# – 32	# – 32
OR	SETCOLOR 1	NONE	# + 64	# – 64	NONE
MODE 2	SETCOLOR 2	# + 160	# + 160	# + 96	# – 96
	SETCOLOR 3	# + 128	# + 192	# + 64	# + 28

2. Luminance controlled by SETCOLOR 1, 0, LUM.

2. Using Table 9-7, locate column 4. Conversion is the character number minus 32 (114 – 32 = 82).
3. **POKE** the Character Base Address (CHBAS) with 226 to specify lower case letters or special graphics characters, e.g.,

   POKE 756,226
   or
   CHBAS = 756
   POKE CHBAS, 226

To return to upper case letters, numbers, and punctuation marks, **POKE** CHBAS with 224.

4. A **PRINT** statement using the converted number (82) assigns the lower case "r" to **SETCOLOR** 0 in mode 2 (see Table 9-5).

Graphic Control Characters

These characters are produced when the CTRL key is pressed with the appropriate alphabetic keys. These characters can be used to draw design, pictures, etc., in mode 0 and in modes 1 and 2 if CHBAS is changed.

COLOR ASSIGNMENT IN THE GTIA MODES 9, 10, and 11:

The GTIA modes 9, 10 and 11 handle colors differently than modes 0 through 8. The following procedures describe how to use modes 9, 10, and 11.

Mode 9: In this mode, one color with 16 luminances is available. First, choose a hue from Table 9-3 and assign it with the **SETCOLOR** command. Only **SETCOLOR** register 4 is used and the luminance must be set to zero; e.g.,

100 **SETCOLOR** 4, HUE, 0 (where HUE is the hue to be assigned)

Then, use the **COLOR** statement to choose luminances from 0 through 15. 0 is almost black and 15 is almost white.

Mode 10: Nine colors with nine different luminances are available. The nine colors are chosen by using **COLOR** 0 through 8. These colors are assigned by use of **POKE** and **SETCOLOR.**

COLOR	POKE location	SETCOLOR register
0	704	---
1	705	---
2	706	---
3	707	---
4	708	0
5	709	1
6	710	2

COLOR	**POKE** location	**SETCOLOR** *(continued)* register
7	711	3
8	712	4

COLORs 4 through 8 can be assigned by using **SETCOLOR** in the normal manner. All **COLOR**s can be assigned by **POKE**ing to the locations given above.

Mode 11: 16 colors, all with the same luminance, are available. The luminance is assigned by **SETCOLOR.** Only **SETCOLOR** register 4 is used with the hue number of zero; e.g.,

100 **SETCOLOR** 4,0,LUM (where lum is the luminance chosen)

The colors are chosen by **COLOR**s 0 to 15. The **COLOR** numbers are the same as those given in Table 9-3.

10 SOUNDS AND GAME CONTROLLERS

This section describes the statement used to generate musical notes and sounds through the audio system of the television monitor. Up to four different sounds can be "played" simultaneously creating harmony. This SOUND statement can also be used to simulate explosions, whistles, and other interesting sound effects. The other commands described in this section deal with the functions used to manipulate the keyboard, joystick, and paddle controllers. These functions allow these controllers to be plugged in and used in BASIC programs for games, etc.

The command and functions covered in this section are:

SOUND PADDLE STICK STRIG PTRIG

SOUND (SO.)

Format: **SOUND** aexp1, aexp2, aexp3, aexp4
Example: 100 **SOUND** 2, 204, 10, 12

The **SOUND** statement causes the specified note to begin playing as soon as the statement is executed. The note will continue playing until the program encounters another **SOUND** statement with the same aexp1 or an **END** statement. This command can be used in either Direct or Deferred modes.

The **SOUND** parameters are described as follows:

aexp1 = *Voice*. Can be 0–3, but each voice requires a separate **SOUND** statement.

aexp2 = *Pitch*. Can be any number between 0–255. The larger the number, the lower the pitch. Table 10-1 defines the pitch numbers for the various musical notes ranging from two octaves above middle C to one octave below middle C.

aexp3 = *Distortion*. Can be even numbers between 0–14. Used in creating sound effects. A 10 is used to create a "pure" tone whereas a 12 gives an interesting buzzer sound. The following program combines the 10 and 12 sounds:

```
10 X = 1
20 X = X+2:? X
30 IF X < 10 THEN GOTO 20
35 SOUND 2, 100, 10, 8
40 Y = 0
50 Y = Y + 2:? Y
60 SOUND 2, 100, 12, 8
70 IF Y < 10 THEN GOTO 50
80 GOTO 10
```

The rest of the numbers are used for other special effects, noise generation, and experimental use.

aexp4 = *Volume control.* Can be between 0 and 15. Using a 1 creates a sound barely audible whereas a 15 is loud. A value of 8 is considered normal. If more than 1 **sound** statement is being used, the total volume should not exceed 32. This creates an unpleasant "clipped" tone. A value of 0 turns off the sound of the specified voice.

Using the note values in Table 10-1, the program in Figure 10-1 demonstrates how to write a program that will "play" the C scale.

TABLE 10-1 TABLE OF PITCH VALUES FOR THE MUSICAL NOTES

HIGH	C	29
NOTES	B	31
	A# or B ♭	33
	A	35
	G# or A ♭	37
	G	40
	F# or G ♭	42
	F	45
	E	47
	D# or E	50
	D	53
	C# or D ♭	57
	C	60
	B	64
	A# or B	68
	A	72
	G# or A ♭	76
	G	81
	F# or G ♭	85
	F	91
	E	96
	D# or E ♭	102
	D	108
	C# or D ♭	114
MIDDLE C	C	121
	B	128
	A# or B ♭	136
	A	144
	G# or A ♭	153
	G	162
	F# G or ♭	173
	F	182
LOW NOTES	E	193
	D# or E ♭	204
	D	217
	C# or D ♭	230
	C	243

```
10 READ A
20 IF A=256 THEN END
30 SOUND 0,A,10,10
40 FOR W=1 TO 400:NEXT W
50 PRINT A
60 GOTO 10
70 END
80 DATA 29,31,35,40,45,47,53,60,64,72,
81,91,96,108,121
90 DATA 128,144,162,182,193,217,243,25
6
```

Figure 10-1. Musical Scale Program

Note that the **DATA** statement in line 90 ends with 256, which is outside of the designated range. The 256 is used as an end-of-data marker.

GAME CONTROLLER FUNCTIONS

Figure 10-2 is an illustration of controllers used with the ATARI Home Computers. The controllers can be attached directly to the ATARI Home Computer or to external mechanical devices so that outside events can be fed directly to the computer for processing and control purposes.

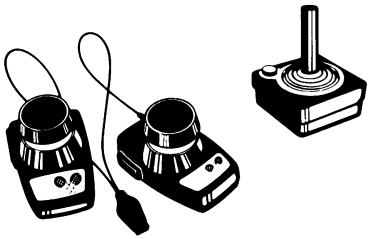


Figure 10-2. Game Controllers

PADDLE

Format: **PADDLE**(aexp)
Example: **PRINT PADDLE**(3)

This function returns the status of a particular numbered controller. The paddle controllers (aexp) are numbered 0-7 from left to right for the ATARI 800 and 400, and 0-3 for the ATARI 1200. This function can be used with other functions or commands to "cause" further actions like sound, graphics controls, etc. for example, the statement IF PADDLE (3) © 14 THEN PRINT "PADDLE ACTIVE." Note that the PADDLE function returns a number between 1 and 228, with the number increasing in size as the knob on the controller is rotated counterclockwise (turned to the left).

PTRIG

Format: **PTRIG**(aexp)
Example: 100 **IF PTRIG**(4) = 0 **THEN PRINT** "MISSILES FIRED!"

The PTRIG function returns a status of 0 if the trigger button of the designated controller is pressed. Otherwise, it returns a value of 1. The aexp must be a number between 0 and 7 for the ATARI 800 and 400, and 0-3 for the ATARI 1200 as it designates the controller.

STICK

Format: **STICK**(aexp)
Example: 100 **PRINT STICK**(3)

This function works exactly the same way as the **PADDLE** command, but can be used with the joystick controller. The joystick controllers are numbered from (left to right) 0-3 for the ATARI 800 and 400, and 0-1 for the ATARI 1200 and 400.

Controller 1 = STICK(0)
Controller 2 = STICK(1)
Controller 3 = STICK(2)
Controller 4 = STICK(3)

Figure 10-3 shows the numbers that are returned when the joystick controller is moved in any direction.

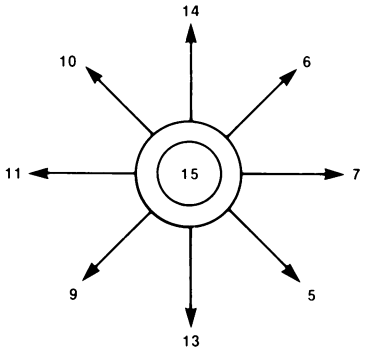


Figure 10-3. Joystick Controller Movement

STRIG

Format: **STRIG**(aexp)
Example: 100 **IF STRIG**(3) = 0 **THEN PRINT** "FIRE TORPEDO"

The **STRIG** function works the same way as the **PTRIG** function. It is used with the joystick. The aexp for the ATARI 800 and 400 must be 0-3, and the aexp for the ATARI 1200 must be 0-1.

11 ADVANCED PROGRAMMING TECHNIQUES

This section includes hints on increasing programming efficiency, conserving memory, and combining machine language programs with ATARI BASIC programs. This section does not include an instruction set for the 6502 microprocessor chip nor does it give instructions on programming in machine language. An additional purchase of the ATARI Assembler Editor cartridge and a careful study of the ATARI Assembler Editor Manual are strongly recommended.

MEMORY CONSERVATION

These hints give ways of conserving memory. Some of these methods make programs less readable and harder to modify, but there are cases where this is necessary due to memory limitations.

1. In many small computers, eliminating blank spaces between words and characters as they are typed into the keyboard will save memory. This is not true of the ATARI Home Computer System, which removes extra spaces. Statements are always displayed the same regardless of how many spaces were used on program entry. Spaces should be used (just as in typing on a conventional typewriter) between successive keywords and between keywords and variable names. Here is an example:

   10 **IF** A = 5 **THEN PRINT** A

   Note the space between **IF** and A and between **THEN** and **PRINT**. In most cases, a statement will be interpreted correctly by the computer even if all spaces are left out, but this is not always true. Use conventional spacing.

2. Each new line number represents the beginning of what is called a new "logical line". Each logical line takes 6 bytes of "overhead", whether it is used to full capacity or not. Adding an additional BASIC statement by using a colon (:) to separate each pair of statements on the same line takes only 3 bytes.
   If you need to save memory, this program:

```
10 X=X+1
20 Y=Y+1
30 Z=X+Y
40 PRINT Z
50 GOTO 50
```

   can be entered on one line:

```
10 X=X+1:Y=Y+1:Z=X+Y:PRINT Z:GOTO 10
```

   This consolidation saves 12 bytes.

3. Variables and constants should be "managed" for savings, too. Each time a constant (4,5,16,3.14159, etc.) is used, it takes 7 bytes. Defining a new variable requires 8 bytes plus the length of the variable name (in characters). But each time it is used after being defined, it takes only 1 byte, regardless of its length. Thus, if a constant (such as 3.14159) is used more than once or twice in a program, it should be defined as a variable, and the variable name used throughout the program. For example:

```
10 PI=3.14159
20 PRINT "AREA OF A CIRCLE IS THE RADI
US SQUARED TIMES ";PI
```

4. Literal strings require 2 bytes overhead and 1 byte for each character (including all spaces) in the string.
5. String variables take 9 bytes each plus the length of the variable name (including spaces) plus the space eaten up by the **DIM** statement plus the size of the string itself (1 byte per character, including spaces) when it is defined. Obviously, the use of string variables is very costly in terms of RAM.
6. Definition of a new matrix requires 15 bytes plus the length of the matrix variable name plus the space needed for the **DIM** statement plus 6 times the size of the matrix (product of the number of rows and the number of columns). Thus, a 25 row by 4 column matrix would require 15 + approximately 3 (for variable name) + approximately 10 (for the **DIM** statement) + 6 times 100 (the matrix size), or about 630 bytes.
7. Each character after **REM** takes one byte of memory. Remarks are helpful to people trying to understand a program, but sometimes it is necessary to remove remark statements to save memory.
8. Subroutines can save memory because one subroutine and several short calls take less memory than duplicating the code several times. On the other hand, a subroutine that is only called once takes extra bytes for the **GOSUB** and RETURN statements.
9. Parentheses take one byte each. Extra parentheses are a good idea in some cases if they make an expression more understandable to the programmer. However, removing unnecessary parentheses and relying on operator precedence will save a few bytes.

PROGRAMMING IN MACHINE LANGUAGE

Machine language is written entirely in binary code. The ATARI Home Computer contains a 6502 microprocessor and it is possible to call 6502 machine code subroutines from BASIC using the USR function. Short routines may then be entered into a program by hand assembly (if necessary).

Before it returns to BASIC, the assembly language routine must do a pull accumulator (**PLA**) instruction to remove the number (N) of input arguments off the stack. If this number is not 0, then all of the input arguments must be **popped** off the stack also using PLA. (See Figure 6-1).

The subroutine should end by placing the low byte of its result in location 212 (decimal), and then return to BASIC using an **RTS** (Return from Subroutine) instruction. The BASIC interpreter will convert the 2-byte binary number stored in locations 212 and 213 into an integer between 0 and 65535 in floating-point format to obtain the value returned by the USR function.

The **ADR** function may be used to pass data that is stored in arrays or strings to a subroutine in machine language. Use the **ADR** function to get the address of the array or string, and then use this address as one of the **USR** input arguments.

The program in Figure 11-1, Hexcode Loader, provides the means of entering hexadecimal codes, converting each hexadecimal number to decimal, and storing the decimal number into an array. The array is then executed as an assembly language subroutine. (An array is used to allocate space in memory for the routine.)

1. To use this program, first enter it. After entering it, save this program on disk or cassette for future use.

```
10 GRAPHICS 0:? "HEXCODE LOADER":?
20 REM STORES DECIMAL EQUIVALENTS IN A
RRAY A, OUTPUTS IN PRINTED 'DATA STATE
MENTS' AT LINE NUMBER 1500.
30 REM USER THEN PLACES CURSOR ON PRIN
TED OUTPUT LINE, HITS "RETURN", AND
40 REM ENTERS REST OF BASIC PROGRAM,IN
CLUDING USER STATEMENT
50 DIM A(50),HEX$(5)
60 REM INPUT,CONVERT,STORE DATA
70 N=0:? "ENTER 1 HEX CODE.  IF LAST O
NE IS IN. ENTER 'DONE'."
80 INPUT HEX$
90 IF HEX$="DONE" THEN N=999:GOTO 140
100 FOR I=1 TO LEN(HEX$)
110 IF HEX$(I,I)<="9" THEN N=N*16+VAL(
HEX$(I,I)):GOTO 130
120 N=N*16+ASC(HEX$(I,I))-ASC("A")+10
130 NEXT I
140 PRINT N:C=C+1
150 A(C)=N
160 IF N<>999 THEN GOTO 70
170 REM PRINTOUT DATA LINE AT 1500
180 GRAPHICS 0:PRINT "1500 DATA ";
190 C=0
200 C=C+1
210 IF A(C)=999 THEN PRINT "999":STOP
220 PRINT A(C);",";
230 A(C)=0
240 GOTO 200
250 PRINT "PUT CORRECT NUMBER OF HEX B
YTES IN LINE 270":STOP :REM TRAP LINE
260 REM **EXECUTION MODULE**
270 CLR :BYTES=0
```

```
280 TRAP 250:DIM E$(1),E(INT(BYTES/6)+
1)
290 FOR I=1 TO BYTES
300 READ A:IF A>255 THEN GOTO 320
310 POKE ADR(E$)+I,A
320 NEXT I
330 REM BASIC PORTION OF USER'S PROGRA
M FOLLOWS:
```

Figure 11-1. Hexcode Loader Input Program

2. Now add the BASIC language part of your program starting at line 340 including the **USR** function that calls the machine language subroutine. (See example below.)
3. Count the total number of hex codes to be entered and enter this number on line 270 when requested. If another number is already entered, simply replace it.
4. Run the program and enter the hexadecimal codes of the machine level subroutine pressing **RETURN** after each entry. After the last entry, type **DONE** and press **RETURN.**
5. Now the **DATA** line (1500) displays on the screen. It will not be entered into the program until the cursor is moved to the **DATA** line and **RETURN** is pressed.
6. Add a program line 5 **GOTO** 270 to bypass the hexcode loader (or delete the hexcode loader through line 260). Now save the completed program by using **CSAVE** or **SAVE**. It is important to do this *before* executing the part of the program containing the **USR** call. A mistake in a machine language routine may cause the system to crash. If the system does hang up, press **SYSTEM RESET.** If the system doesn't respond, turn power off and on again, reload the program, and correct it.

Note: This method only works with *relocatable* machine language routines.

The following two sample programs can each be entered into the Hexcode Loader program. The first program prints NOTHING IS MOVING while the machine program changes the colors. Use inverse video for lines 380 and 390. The second sample program displays a BASIC graphics design, then changes colors.

```
340 GRAPHICS 1+16
350 FOR I=1 TO 6
360 PRINT #6;" NOTHING IS MOVING"
370 PRINT #6;" nothing is moving"
380 PRINT #6;" NOTHING IS MOVING"
390 PRINT #6;" nothing is moving"
400 NEXT I
410 Q=USR(ADR(E$)+1)
420 FOR I=1 TO 25:NEXT I:GOTO 410
```

After entering this program, check that line 270 reads:

270 **CLR:**BYTES = 21

Type **RUN** and press **RETURN.**

Now enter the hexadecimal codes as shown column by column.

```
68    2
A2    E8
0     E0
AC    3
C4    90
2     F5
BD    8C
C5    C7
2     2
9D    60
C4
```

BYTES = 21

When completed, type **DONE** and press **RETURN.** Now place the cursor after the last entry (999) on the **DATA** line and press **RETURN.**

Now run the program by typing **GOTO** 270 and pressing **RETURN,** or add line 5 has been added, type **RUN** and press **RETURN.** Press **BREAK** to stop program.

The second program, which follows, should be entered in place of the NOTHING IS MOVING program. Be sure to check the BYTES = _____ count in line 270. Delete line 5. Follow steps 2 through 6.

```
340 GRAPHICS 7+16
350 SETCOLOR 0,9,4
360 SETCOLOR 1,9,6
370 SETCOLOR 2,9,8
380 CR=1
390 FOR X=0 TO 159
400 COLOR INT(CR)
410 PLOT 80,0
420 DRAWTO X,95
430 CR=CR+0.125
440 IF CR=4 THEN CR=1
450 NEXT X
460 X=USR(ADR(E$)+1)
470 FOR I=1 TO 15:NEXT I
480 GOTO 460
```

Type **RUN** and press **RETURN.**

Enter the hexadecimal codes for this program column by column.

```
68    2
A2    E8
0     E0
AC    2
C4    90
2     F5
BD    8C
```

C5 C6
2 2
9D 60
C4

BYTES = 21

When completed, type **DONE** and press **RETURN.** Now place the cursor after the last entry (999) on the **DATA** line and press **RETURN.**

Now run the program by typing **GOTO** 270 and pressing **RETURN,** or add line 5 **GOTO** 270 and type **RUN** and press **RETURN.** Press **BREAK** to stop the program. To use the Hexcode loader for other programs, be sure to delete line 5.

Figure 11-2 illustrates an assembler subroutine used to rotate colors which might prove useful. It is included here for the information of the user.

Assembler Subroutine to Rotate Colors

Address	Object Code	Line No.	Label	Mnemonic	Data	
		0100				Routine to rotate COLOR data
		0110				From one register to another.
		0120				4 colors are rotated.
		0130				
		0140				Operating system address
02C4		0150				COLOR 0 = $02C4
02C5		0160				COLOR 1 = $02C5
02C6		0170				COLOR 2 = $02C6
02C7		0175				COLOR 3 = $02C7
		0180				
		0190		* =	$6000	Machine program starting address*
6000	68	0200		PLA		Pop stack (See Chapter 4)
6001	A200	0210		LDX	#0	Zero the X register
6003	ACC402	0220		LDY	COLOR0	Save COLOR 0
6006	BDC502	0230	LOOP	LDA	COLOR1,X	
6009	9DC402	0240		STA	COLOR0,X	
600C	E8	0250		INX		Increment the X register (add one)
600D	E003	0260		CPX	#3	Compare contents of X register with 3
600F	90F5	0270		BCC	LOOP	Loop if X register contents are less than 3
6011	8CC702	0280		STY	COLOR3	Save COLOR 0 in COLOR 3
6014	60	0290		RTS		Return from machine level subroutine

Assembler Prints This — This Portion is Source Information Programmer Enters Using ATARI Assembler Editor Cartridge

# Indicates data (source)
* Routine is relocatable
$ Indicates a hexadecimal number

Figure 11-2. Assembler Subroutine To Rotate Colors

A APPENDIX ALPHABETICAL DIRECTORY OF BASIC RESERVED WORDS

Note: The period is mandatory after all abbreviated keywords.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
ABS		Function returns absolute value (unsigned) of the variable or expression.
ADR		Function returns memory address of a string.
AND		Logical operator: Expression is true only if both subexpressions joined by **AND** are true.
ASC		String function returns the numeric value of a single string character.
ATN		Function returns the arctangent of a number or expression in radians or degrees.
BYE	**B.**	Exit from BASIC and return to the resident operating system or console processor.
CLOAD	**CLOA.**	Loads data from Program Recorder into RAM.
CHR$		String function returns a single string byte equivalent to a numeric value between 0 and 255 in ATASCII code.
CLOG		Function returns the base 10 logarithm of an expression.
CLOSE	**CL.**	I/O statement used to close a file at the conclusion of I/O operations.
CLR		The opposite of DIM: Undimensions all strings and arrays.
COLOR	**C.**	Chooses color register to be used in color graphics work.
CONT	**CON.**	Continue. Causes a program to restart execution on the next line following use of the **BREAK** key or encountering a **STOP.**
COS		Function returns the cosine of the variable or expression (degrees or radians).
CSAVE		Outputs data from RAM to the Program Recorder for tape storage.
DATA	**D.**	Part of **READ/DATA** combination. Used to identify the succeeding items (which must be separated by commas) as individual data items.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
DEG	**DE.**	Statement **DEG** tells computer to perform trigonometric functions in degrees instead of radians. (Default in radians.)
DIM	**DI.**	Reserves the specified amount of memory for matrix, array, or string. All string variables, arrays, matrices must be dimensioned with a DIM statement.
DOS	**DO.**	Reserved word for disk operators. Causes the menu to be displayed. (See *DOS Manual*.)
DRAWTO	**DR.**	Draws a straight line between a plotted point and specified point.
END		Stops program execution; closes files; turns off sounds. Program may be restarted using **CONT.** (Note: **END** may be used more than once in a program.)
ENTER	**E.**	I/O command used to store data or programs in untokenized (source) form.
EXP		Function returns e (2.7182818) raised to the specified power.
FOR	**F.**	Used with **NEXT** to establish **FOR/NEXT** loops. Introduces the range that the loop variable will operate in during the execution of loop.
FRE		Function returns the amount of remaining user memory (in bytes).
GET	**GE.**	Used mostly with disk operations to input a single byte of data.
GOSUB	**GOS.**	Branch to a subroutine beginning at the specified line number.
GOTO	**G.**	Unconditional branch to a specified line number.
GRAPHICS	**GR.**	Specifies which of the graphics modes is to be used. **GR.0** may be used to clear screen.
IF		Used to cause conditional branching or to execute another statement on the same line (only if the first expression is true).
INPUT	**I.**	Causes computer to ask for input from keyboard. Execution continues only when **RETURN** key is pressed after inputting data.
INT		Function returns the next lowest whole integer below the specified value. Rounding is always downward, even when number is negative.
LEN		String function returns the length of the specified string in bytes or characters (1 byte contains 1 character).

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
LET	**LE.**	Assigns a value to a specific variable name. LET is optional in ATARI BASIC, and may be simply omitted.
LIST	**L.**	Display or otherwise output the program list.
LOAD	**LO.**	Input from disk, etc. into the computer.
LOCATE	**LOC.**	Graphics: Stores, in a specified variable, the value that controls a specified graphics point.
LOG		Function returns the natural logarithm of a number.
LPRINT	**LP.**	Command to line printer to print the specified message.
NEW		Erases all contents of user RAM.
NEXT	**N.**	Causes a **FOR/NEXT** loop to terminate or continue depending on the particular variables or expressions. All loops are executed at least once.
NOT		A "1" is returned only if the expression is NOT true. If it is true, a "0" is returned.
NOTE	**NO.**	See *DOS/FMS Manual*...used only in disk operations.
ON		Used with **GOTO** or **GOSUB** for branching purposes. Multiple branches to different line numbers are possible depending on the value of the **ON** variable or expression.
OPEN	**O.**	Opens the specified file for input or output operations.
OR		Logical operator used between two expressions. If either one is true, a "1" is evaluated. A "0" results only if both are false.
PADDLE		Function returns position of the paddle game controller.
PEEK		Function returns decimal form of contents of specified memory location (RAM or ROM).
PLOT	**PL.**	Causes a single point to be plotted at the X,Y location specified.
POINT	**P.**	See DOS/FMS Manual...used only in disk operations.
POKE	**POK.**	Insert the specified byte into the specified memory location. May be used only with RAM. Don't try to **POKE** ROM or you'll get an error.
POP		Removes the loop variable from the **GOSUB** stack. Used when departure from the loop is made in other than normal manner.
POSITION	**POS.**	Sets the cursor to the specified screen position.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
PRINT	**PR.** or **?**	I/O command causes output from the computer to the specified output device.
PTRIG		Function returns status of the trigger button on paddle game controllers.
PUT	**PU.**	Causes output of a single byte of data from the computer to the specified device.
RAD		Specifies that information is in radians rather than degrees when using the trigonometric functions. Default is to **RAD.** (See **DEG.**)
READ	**REA.**	Read the next items in the **DATA** list and assign to specified variables.
REM	**R.** or (SPACE).	Remarks. This statement does nothing, but comments may be printed within the program list for future reference by the programmer. Statements on a line that starts with **REM** are not executed.
RESTORE	**RES.**	Allows **DATA** to be **read** more than once.
RETURN	**RET.**	**RETURN** from subroutine to the statement immediately following the one in which **GOSUB** appeared.
RND		Function returns a random number between 0 and 1, but never 1.
RUN	**RU.**	Execute the program. Sets normal variables to 0, undims arrays and string.
SAVE	**S.**	I/O statement causes data or program to be recorded on disk under filespec provided with **SAVE.**
SETCOLOR	**SE.**	Store hue and luminance color data in a particular color register.
SGN		Function returns + 1 if value is positive, 0 if zero, − 1 if negative.
SIN		Function returns trigonometric sine of given value (**DEG** or **RAD**).
SOUND	**SO.**	Controls register, sound pitch, distortion, and volume of a tone or note.
SQR		Function returns the square root of the specified value.
STATUS	**ST.**	Calls status routine for specified device.
STEP		Used with **FOR/NEXT.** Determines increment to be skipped between each pair of loop variable values.
STICK		Function returns position of stick game controller.

RESERVED WORD:	ABBREVIATION:	BRIEF SUMMARY OF BASIC STATEMENT
STRIG		Function returns 1 if stick trigger button not pressed, 0 if pressed.
STOP	**STO.**	Causes execution to stop, but does not close files or turn off sounds.
STR$		Function returns a character string equal to numeric value given. For example: **STR$(65)** returns 65 as a string.
THEN		Used with **IF** statement. If expression is true, the **THEN** statements are executed. If the expression is false, control passes to next line.
TO		Used with **FOR** as in "**FOR** X = 1 **TO** 10". Separates the loop range expressions.
TRAP	**T.**	Takes control of program in case of an **INPUT** error and directs execution to a specified line number.
USR		Function returns results of a machine-language subroutine.
VAL		Function returns the equivalent numeric value of a string.
XIO	**X.**	General I/O statement used with disk operations (see *DOS/FMS Manual)* and in graphics work (Fill).

ERROR MESSAGES

APPENDIX B

ERROR CODE NO.	ERROR CODE MESSAGE
2	**Memory insufficient** to store the statement or the new variable name or to **DIM** a new string variable.
3	**Value Error:** A value expected to be a positive integer is negative, a value expected to be within a specific range is not.
4	**Too Many Variables:** A maximum of 128 different variable names is allowed. (See **Variable Name Limit**.)
5	**String Length Error:** Attempted to store beyond the **DIM**ensioned string length.
6	**Out of Data Error: READ** statement requires more data items than supplied by **DATA** statement(s).
7	**Number greater than 32767:** Value is not a positive integer or is greater than 32767.
8	**Input Statement Error:** Attempted to **INPUT** a non-numeric value into a numeric variable.
9	**Array or String DIM Error: DIM** size is greater than 32767 or an array/matrix reference is out of the range of the dimensioned size, or the array/matrix or string has been already **DIM**ensioned, or a reference has been made to an undimensioned array or string.
10	**Argument Stack Overflow:** There are too many **GOSUB**s or too large an expression.
11	**Floating Point Overflow/Underflow Error:** Attempted to divide by zero or refer to a number larger than 1×10^{98} or smaller than 1×10^{-99}.
12	**Line Not Found:** A **GOSUB**, **GOTO**, or **THEN** referenced a non-existent line number.
13	**No Matching FOR Statement:** A **NEXT** was encountered without a previous **FOR** or nested **FOR/NEXT** statements do not match properly. (Error is reported at the **NEXT** statement, not at **FOR**).
14	**Line Too Long Error:** The statement is too complex or too long for BASIC to handle.
15	**GOSUB** or **FOR** Line Deleted: A **NEXT** or **RETURN** statement was encountered and the corresponding **FOR** or **GOSUB** has been deleted since the last **RUN**.
16	**RETURN Error:** A **RETURN** was encountered without a matching **GOSUB**.

ERROR CODE NO.	ERROR CODE MESSAGE
17	**Garbage Error:** Execution of "garbage" (bad RAM bits) was attempted. This error code may indicate a hardware problem, but may also be the result of faulty use of **POKE**. Try typing **NEW** or powering down, then re-enter the program without any **POKE** commands.
18	**Invalid String Character:** String does not start with a valid character, or string in **VAL** statement is not a numeric string.
Note:	**The following are INPUT/OUTPUT errors that result during the use of disk drives, printers, or other accessory devices. Further information is provided with the auxiliary hardware.**
19	**LOAD program Too Long:** Insufficient memory remains to complete **LOAD**.
20	**Device Number Larger** than 7 or Equal to 0.
21	**LOAD File Error:** Attempted to **LOAD** a non-**LOAD** file.
128	**BREAK Abort:** User hit BREAK key during I/O operation.
129	**IOCB**[1] already open.
130	**Nonexistent Device** specified.
131	**IOCB Write Only. READ** command to a write-only device (Printer).
132	**Invalid Command:** The command is invalid for this device.
133	**Device or File not Open:** No **OPEN** specified for the device.
134	**Bad IOCB Number:** Illegal device number.
135	**IOCB Read Only Error: WRITE** command to a read-only device.
136	**EOF:** End of File read has been reached. (**NOTE:** This message may occur when using cassette files.)
137	**Truncated Record:** Attempt to read a record longer than 256 characters.
138	**Device Timeout.** Device doesn't respond.
139	**Device NAK:** Garbage at serial port or bad disk drive.
140	**Serial bus** input framing error.
141	**Cursor out of range** for particular mode.
142	**Serial bus data frame overrun.**
143	**Serial bus data frame checksum error.**
144	**Device done error** (invalid "done" byte): Attempt to write on a write-protected diskette or a bad sector.
145	**BAD screen mode error.**
146	**Function not implemented** in handler.
147	**Insufficient RAM** for operating selected graphics mode.
160	**Drive number error.**
161	**Too many OPEN files** (no sector buffer available).

[1]IOCB refers to Input/Output Control Block.

ERROR CODE NO.	ERROR CODE MESSAGE
162	**Disk full** (no free sectors).
163	**Unrecoverable system data I/O error.**
164	**File number mismatch:** Links on disk are messed up.
165	**File name error.**
166	**POINT data length error.**
167	**File locked.**
168	**Command invalid** (special operation code).
169	**Directory full** (64 files).
170	**File not found.**
171	**POINT invalid.**

C APPENDIX

ATASCII CHARACTER SET

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
0	0		á	13	D		ú
1	1		ù	14	E		ó
2	2		Ñ	15	F		ö
3	3		É	16	10		ü
4	4		Ç	17	11		â
5	5		ô	18	12		û
6	6		ò	19	13		î
7	7		ì	20	14		é
8	8		£	21	15		è
9	9		ï	22	16		ñ
10	A		ü	23	17		ê
11	B		ä	24	18		å
12	C		Ö	25	19		à

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
26	1A		Å	42	2A	*	
27	1B			43	2B	+	
28	1C			44	2C	,	
29	1D			45	2D	-	
30	1E			46	2E		
31	1F			47	2F	/	
32	20	Space		48	30	0	
33	21	!		49	31	1	
34	22	''		50	32	2	
35	23	#		51	33	3	
36	24	$		52	34	4	
37	25	%		53	35	5	
38	26	&		54	36	6	
39	27	'		55	37	7	
40	28	(		56	38	8	
41	29	)		57	39	9	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
58	3A			74	4A	J	
59	3B	;		75	4B	K	
60	3C	<		76	4C	L	
61	3D	=		77	4D	M	
62	3E	>		78	4E	N	
63	3F	?		79	4F	O	
64	40	@		80	50	P	
65	41	A		81	51	Q	
66	42	B		82	52	R	
67	43	C		83	53	S	
68	44	D		84	54	T	
69	45	E		85	55	U	
70	46	F		86	56	V	
71	47	G		87	57	W	
72	48	H		88	58	X	
73	49	I		89	59	Y	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
90	5A	Z		106	6A	j	
91	5B	[		107	6B	k	
92	5C	\		108	6C	l	
93	5D	]		109	6D	m	
94	5E	^		110	6E	n	
95	5F	—		111	6F	o	
96	60		i	112	70	p	
97	61	a		113	71	q	
98	62	b		114	72	r	
99	63	c		115	73	s	
100	64	d		116	74	t	
101	65	e		117	75	u	
102	66	f		118	76	v	
103	67	g		119	77	w	
104	68	h		120	78	x	
105	69	i		121	79	y	

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
122	7A	z		138	8A		
123	7B		Ä	139	8B		
124	7C	|		140	8C		
125	7D			141	8D		
126	7E			142	8E		
127	7F			143	8F		
128	80			144	90		
129	81			145	91		
130	82			146	92		
131	83			147	93		
132	84			148	94		
133	85			149	95		
134	86			150	96		
135	87			151	97		
136	88			152	98		
137	89			153	99		

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
154	9A			170	AA		
155	9B	(EOL) RETURN		171	AB		
156	9C			172	AC		
157	9D			173	AD		
158	9E			174	AE		
159	9F			175	AF		
160	A0			176	B0		
161	A1			177	B1		
162	A2			178	B2		
163	A3			179	B3		
164	A4			180	B4		
165	A5			181	B5		
166	A6			182	B6		
167	A7			183	B7		
168	A8			184	B8		
169	A9			185	B9		

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
186	BA			202	CA		
187	BB			203	CB		
188	BC			204	CC		
189	BD			205	CD		
190	BE			206	CE		
191	BF			207	CF		
192	C0			208	D0		
193	C1			209	D1		
194	C2			210	D2		
195	C3			211	D3		
196	C4			212	D4		
197	C5			213	D5		
198	C6			214	D6		
199	C7			215	D7		
200	C8			216	D8		
201	C9			217	D9		

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
218	DA			234	EA		
219	DB			235	EB		
220	DC			236	EC		
221	DD			237	ED		
222	DE			238	EE		
223	DF			239	EF		
224	E0			240	F0		
225	E1			241	F1		
226	E2			242	F2		
227	E3			243	F3		
228	E4			244	F4		
229	E5			245	F5		
230	E6			246	F6		
231	E7			247	F7		
232	E8			248	F8		
233	E9			249	F9		

DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER	DECIMAL CODE	HEXADECIMAL CODE	CHARACTER	EUROPEAN CHARACTER
250	FA			253	FD		*(Buzzer)*
251	FB			254	FE		*(Delete character)*
252	FC			255	FF		*(Insert character)*

See Appendix H for a user program that performs decimal/hexadecimal conversion.

Notes:

1. ATASCII stands for "ATARI ASCII". Letters and numbers have the same values as those in ASCII, but some of the special characters are different.
2. Except as shown, characters from 128-255 are reverse colors of 1 to 127.
3. Add 32 to upper case code to get lower case code for same letter.
4. To get ATASCII code, tell computer (direct mode) to PRINT ASC ("_____") Fill blank with letter, character, or number of code. Must use the quotes!
5. The normal display keycaps are shown as white symbols on a black background; the inverse keycap symbols are shown as black on a white background.

HOME COMPUTER MEMORY MAP

APPENDIX D

ATARI 400/800

ADDRESS		CONTENTS
Decimal	**Hexadecimal**	
65535	FFFF	OPERATING SYSTEM ROM
57344	E000	
57343	DFFF	FLOATING POINT ROM
55296	D800	
55295	D7FF	HARDWARE REGISTERS
53248	D000	
53247	CFFF	NOT USED
49152	C000	
49151	BFFF	CARTRIDGE SLOT A
40960	A000	(may be RAM if no A or B cartridge)
40959	9FFF	CARTRIDGE SLOT B
32768	8000	(may be RAM if no B cartridge) ← *RAMTOP (MSB)*
32767	7FFF	(7FFF if 32K system) DISPLAY DATA (size varies)
31755	7C0B	DISPLAY LIST (size varies 7C0B if 32K system) (**GRAPHICS** 0) ← *OS MEMTOP*
		FREE RAM (size varies) ← *BASIC MEMTOP*
10880	2A80	BASIC program, buffers, tables, run-time stack. (2A80 if **DOS**, may vary) ← *OS MEMLO*, *BASIC LOMEM*
10879	2A7F	**DISK OPERATING SYSTEM (2A7F-700)**
9856	2680	DISK I/O BUFFERS (current **DOS**)
9855	267F	DISK OPERATING SYSTEM RAM (current **DOS**)
4864	1300	

ADDRESS		CONTENTS
Decimal	**Hexadecimal**	
4863	12FF	FILE MANAGEMENT SYSTEM RAM (current DOS)
1792	700	
1791	6FF	FREE RAM
1536	600	
1535	5FF	FLOATING POINT (used by BASIC)
1406	57E	
1405	57D	BASIC CARTRIDGE
1152	480	
1151	47F	**OPERATING SYSTEM RAM (47F-200)**
1021	3FD	CASSETTE BUFFER
1020	3FC	RESERVED
1000	3E8	
999	3E7	PRINTER BUFFER
960	3C0	
959	3BF	IOCB's
832	340	
831	33F	MISCELLANEOUS OS VARIABLES
512	200	
511	1FF	HARDWARE STACK
256	100	
255	FF	**PAGE ZERO**
212	D4	FLOATING POINT (used by BASIC)
211	D3	BASIC or CARTRIDGE PROGRAM
210	D2	
209	D1	FREE BASIC RAM
208	D0	
207	CF	FREE BASIC AND ASSEMBLER RAM
203	CB	
202	CA	FREE ASSEMBLER RAM } BASIC ZERO PAGE
176	B0	
128	80	ASSEMBLER ZERO PAGE } BASIC ZERO PAGE
127	7F	OPERATING SYSTEM RAM
0	0	

As the addresses for the top of RAM, OS, and BASIC and the ends of OS and BASIC vary according to the amount of memory, these addresses are indicated by pointers. The pointer addresses for each are defined in Appendix I.

ATARI 1200XL

ADDRESS		CONTENTS
Decimal	**Hexadecimal**	
65535 55296	FFFF D800	OPERATING SYSTEM ROM (or RAM if OS ROM is disabled) See Note 1.
55295 53248	D7FF D000	OS ROM self-test code can only be accessed during self test. Space shared with I/O (PIA, POKEY, ANTIC, GTIA) See Note 3.
53247 49152	CFFF C000	OS ROM (or RAM if OS ROM is disabled) See Note 1.
49151 40960	BFFF A000	CARTRIDGE INTERFACE ROM (may be RAM if no cartridge)
40959 32768	9FFF 8000	CARTRIDGE INTERFACE ROM (may be RAM if no cartridge)
32767 22528	7FFF 5800	RAM SPACE
22527 20480	57FF 5000	RAM (unless in self-test mode) See Note 2.
20479 0000	4FFF 0000	RAM SPACE

Notes:

1. Disable OS ROM by writing a 0 to PB0 of PIA.
2. Self-test OS ROM code accessed at hex address 5000 (if PB0 set to 0) during self test. RAM between 5000 and 57FF cannot be accessed.
3. PIA, POKEY, ANTIC, GTIA registers used as in Atari 400/800 Home Computer.

E APPENDIX DERIVED FUNCTIONS

Derived Functions	Derived Functions in Terms of ATARI Functions
Secant	SEC(X) = 1/COS(X)
Cosecant	CSC(X) = 1/SIN(X)
Inverse Sine	ARCSIN(X) = ATN(X/SQR(-X*X + 1))
Inverse Cosine	ARCCOS(X) = -ATN(X/SQR(-X*X + 1) + CONSTANT)
Inverse Secant	ARSEC(X) = ATN(SQR(X*X-1)) + (SGN(X-1) *CONSTANT)
Inverse Cosecant	ARCCSC(X) = ATN(1/SQR(X*X-1)) + (SGN(X-1) *CONSTANT)
Inverse Cotangent	ARCCOT(X) = ATN(X) + CONSTANT
Hyperbolic Sine	SINH(X) = (EXP(X)-EXP(-X))/2
Hyperbolic Cosine	COSH(X) = (EXP(X) + EXP(-X))/2
Hyperbolic Tangent	TANH(X) = -EXP(-X)/(EXP(X) + EXP(-X))*2 + 1
Hyperbolic Secant	SECH(X) = 2/(EXP(X) + EXP(-X))
Hyperbolic Cosecant	CSCH(X) = 2/(EXP(X)-EXP(-X))
Hyperbolic Cotangent	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2 + 1
Inverse Hyperbolic Sine	ARCSINH(X) = LOG(X + SQR(X*X + 1))
Inverse Hyperbolic Cosine	ARCCOSH(X) = LOG(X + SQR(X*X-1))
Inverse Hyperbolic Tangent	ARCTANH(X) = LOG((1 + X)/(1-X))/2
Inverse Hyperbolic Secant	ARCSECH(X) = LOG((SQR(-X*X + 1) + 1)/X)
Inverse Hyperbolic Cosecant	ARCCSCH(X) = LOG((SGN(X)*SQR(X*X + 1) + 1)/X)
Inverse Hyperbolic Cotangent	ARCCOTH(X) = LOG((X + 1)/(X-1))/2

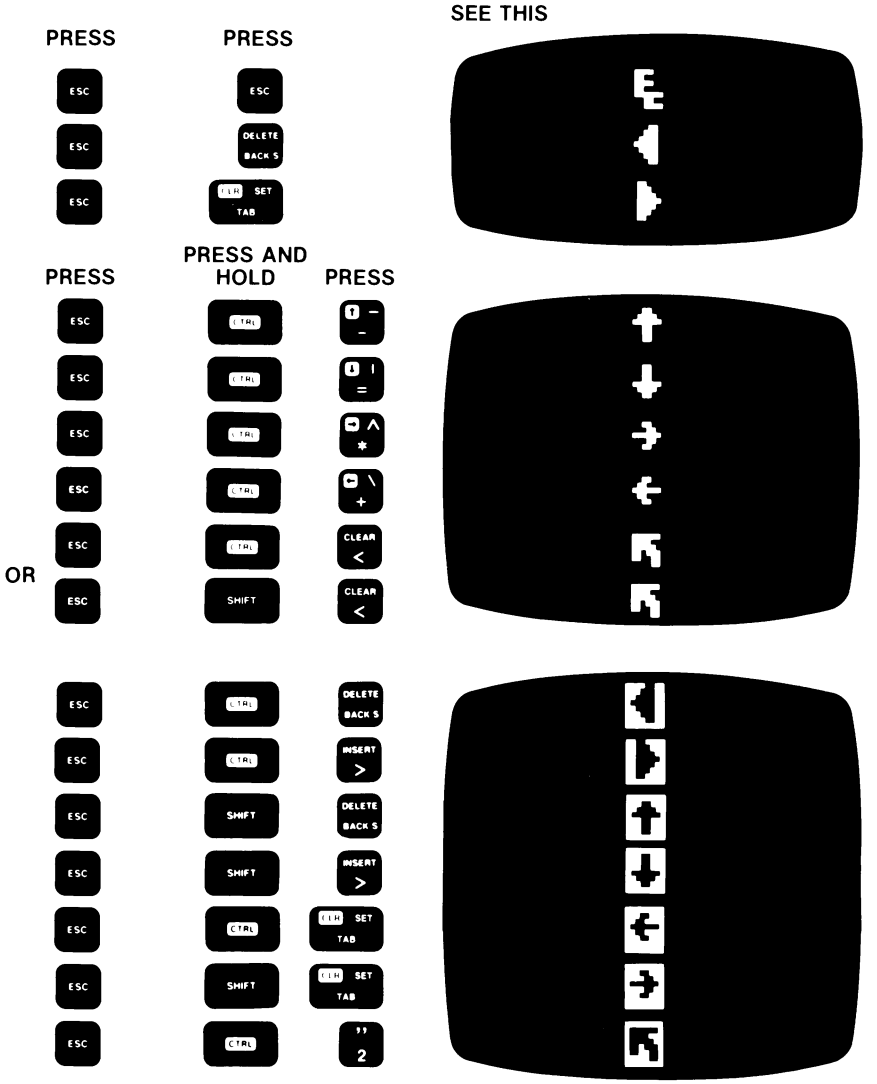
Notes:

1. If in RAD (default) mode, CONSTANT = 1.57079633
   If in DEG mode, CONSTANT = 90.
2. In this chart, the variable X in parentheses represents the value or expression to be evaluated by the derived function. Obviously, any variable name is permissible, as long as it represents the number or expression to be evaluated.

PRINTED VERSIONS OF CONTROL CHARACTERS

APPENDIX F

The cursor and screen control characters can be placed in a string in a program or used as a Direct mode statement by pressing the ESC key before entering the character from the keyboard. This causes the special symbols which are shown below to be displayed. (Refer to Section 1 -ESC Key.)



G APPENDIX GLOSSARY

Alphanumeric: The alphabetic letters A-Z, and the numbers 0-9. (No punctuation marks or graphics symbols).

Array: A list of numerical values stored in a series of memory locations preceded by a DIM statement. May be referred to by use of an array variable, and its individual elements are referred to by subscripted variable names.

ATASCII: Stands for ATARI American Standard Code for Information Interchange.

BASIC: High level programming language. Acronym for Beginner's All-purpose Symbolic Instruction Code. BASIC is always written using all capital letters. Developed by Mssrs. Kemeny and Kurtz at Dartmouth College in 1963.

Binary: A number system using the base two. Thus the only possible digits are 0 and 1, which may be used in a computer to represent true and false, on and off, etc.

Bit: Short for Binary Digit. A bit can be thought of as representing true or false, whether a circuit is on or off, or any other type of two-possibility concept. A bit is the smallest unit of data with which a computer can work.

Branch: ATARI BASIC executes a program in order of line numbers. This execution sequence can be altered by the programmer, and the program can be told to skip over a certain number of lines or return to a line earlier in the program. This contrived change in execution sequence is called "branching".

Bug: A mistake or error usually in the program or "software".

Byte: Usually eight bits (enough to represent the decimal number 255 or 11111111 in binary notation). A byte of data can be used to represent an ATASCII character or a number in the range of 0 to 255.

Central Processing Unit (CPU): In microcomputers such as the ATARI systems, these are also called microprocessors or MPU. At one time, the CPU was that portion of any computer that controlled the memory and peripherals. Now the CPU or MPU is usually found on a single integrated circuit or "chip" (ATARI uses a 6502 microprocessor chip).

Code: Instructions written in a language understood by a computer.

Command: An instruction to the computer that is executed immediately. A good example is the BASIC command **RUN**. (See **Statement**.)

Computer: Any device that can receive and then follow instructions to manipulate information. Both the instructions and the information may be varied from moment to moment. The distinction between a computer and a programmable calculator lies in the computer's ability to manipulate text as well as numbers. Most calculators can only handle numbers.

Concatenation: The process of joining two or more strings together to form one longer string.

Control Characters: Characters produced by holding down the key labeled CTRL while simultaneously pressing another key.

CRT: Abbreviation for "cathode ray tube" (the tube used in a TV set). In practice, this is often used to describe the television receiver used to display computer output. Also called a "monitor."

Cursor: A square displayed on the TV monitor that shows where the next typed character will be displayed.

Data: Information of any kind.

Debug: The process of locating and correcting mistakes and errors in a program.

Default: A mode or condition "assumed" by the computer until it is told to do something else. For example, it will "default" to screen and keyboard unless told to use other I/O devices.

Digital: Information that can be represented by a collection of bits. Virtually all modern computers, especially microcomputers, use the digital approach.

Diskette: A small disk. A record/playback medium like tape, but made in the shape of a flat disk that is placed inside a stiff envelope for protection. The advantage of the disk over cassette or other tape for memory storage is that access to any part of the disk is virtually immediate. The ATARI Home Computer System can control up to 4 diskette drive peripherals simultaneously. In this manual, disk and diskette are used interchangeably.

DOS: Abbreviation for "disk operating system". The software or programs which facilitate use of a disk-drive system. DOS is pronounced either "dee oh ess" or "doss."

Editing: Making corrections or changes in a program or data.

Execute: To do what a command or program specifies. To **RUN** a program or portion thereof.

Expression: A combination of variables, numbers, and operators (like +, −, etc.) that can be evaluated to a single quantity. The quantity may be a string or a number.

Format: To specify the form in which something is to appear.

Hard Copy: Printed output as opposed to temporary TV monitor display.

Hardware: The physical apparatus and electronics that make up a computer.

Increment: Increase in value (usually) by adding one. Used for counting (as in counting the number of repetitions through a loop).

Initialize: Set to an initial or starting value. In ATARI BASIC, all non-array variables are initialized to zero when the command **RUN** is given. Array and string elements are not initialized.

Input: Information transfer to the computer. Output is information transfer away from the computer. In this manual, input and output are always in relation to the computer.

Interactive: A system that responds quickly to the user, usually within a second or two. All home computer systems are interactive.

Interface: The electronics used to allow two devices to communicate.

IOCB Input/Output Control Block. A block of data in RAM that tells the Operating System the information it needs to know for an I/O operation.

I/O: Short for input/output, I/O devices include the keyboard, TV monitor, program recorder, printer, and disk drives.

K: Stands for "kilo" meaning "times 1000". Thus 1 KByte is (approximately) 1000 bytes. (Actually 1024 bytes.) Also, the device type code for the Keyboard.

Keyword: A word that has meaning as an instruction or command in a computer language, and thus must not be used as a variable name or at the beginning of a variable name.

Language: A set of conventions specifying how to tell a computer what to do.

Memory: The part of a computer (usually RAM or ROM) that stores data or information.

Menu: A list of options from which the user may choose.

Microcomputer: A computer based on a microprocessor chip; ATARI uses the 6502.

Monitor: The television receiver used to display computer output.

Null String: A string consisting of no characters whatever.

OS: Abbreviation for Operating System. This is actually a collection of programs to aid the user in controlling the computer. Pronounced "oh ess".

Output: See **input** and **I/O.**

Parallel: Two or more things happening simultaneously. A parallel interface, for example, controls a number of distinct electrical signals at the same time. Opposite of serial.

Peripheral: An I/O device. See **I/O.**

Pixel: Picture Element. One point on the screen display. Size depends on graphics mode being used.

Precedence: Rules that determine the priority in which operations are conducted, especially with regard to the arithmetical/logical operators.

Program: A sequence of instructions that describes a process. A program must be in the language that the particular computer can understand.

Prompt: A symbol that appears on the monitor screen that indicates the computer is ready to accept keyboard input. In ATARI BASIC, this takes the form of the word "READY". A "?" is also used to prompt a user to enter (input) information or take other appropriate action.

RAM: Random Access Memory. The main memory in most computers. RAM is used to store both programs and data.

Random Number Generator: May be hardware (as is ATARI's) or a program that provides a number whose value is difficult to predict. Used primarily for decision-making in game programs, etc.

Reserved Word: See **Keyword.**

ROM: Read Only Memory. In this type of solid-state electronic memory, information is stored by the manufacturer and it cannot be changed by the user. Programs such as the BASIC interpreter and other cartridges used with the ATARI systems use ROM.

Save: To copy a program or data into some location other than RAM (for example, diskette or tape).

Screen: The TV screen. In ATARI BASIC, a particular I/O device code "S:"

Serial: The opposite of parallel. Things happening only one at a time in sequence. Example: A serial interface.

Software: As opposed to Hardware. Refers to programs and data.

Special Character: A character that can be displayed by a computer but is neither a letter nor a numeral. The ATARI graphics symbols are special characters. So are punctuation marks, etc.

Statement: An instruction to the computer. See also **Command.** While all commands may be considered statements, all statements are certainly not commands. A statement contains a line number (deferred mode), a keyword, the value to be operated on, and is terminated by pressing the RETURN key.

String: A sequence of letters, numerals, and other characters. May be stored in a string variable. The string variable's name must end with a $.

Subroutine: A part of a program than can be executed by a special statement (**GOSUB**) in BASIC: This effectively gives a single statement the power of a whole program. The subroutine is a very powerful construct.

Variable: A variable may be thought of as a box in which a value may be stored. Such values are typically numbers and strings.

Window: A portion of the TV display devoted to a specific purpose such as for graphics or text.

H APPENDIX USER PROGRAMS

This appendix contains programs and routines that demonstrate the diverse capabilities of the ATARI Home Computer System. Included in this appendix is a Decimal/Hexadecimal program for those users who write programs that require this type of conversion.

CHECKBOOK BALANCER

This is one of the "traditional" programs that every beginning computerist writes. It allows entry of outstanding checks and uncredited deposits as well as cleared checks and credited deposits.

```
10 DIM A$(30),MSG$(40),MSG1$(30),MSG2$
(30),MSG3$(30),MSG4$(30),MSG5$(30),MSG
6$(30)
20 OUTSTAND=0
30 GRAPHICS 0:? :? "CHECKBOOK BALANCER
":?
40 ? "YOU MAY MAKE CORRECTIONS AT ANY
TIME BY ENTERING A NEGATIVE DOLLAR VAL
UE."
50 MSG1$="OLD CHECK OUTSTANDING"
60 MSG2$="OLD DEPOSIT--NOT CREDITED"
70 MSG3$="OLD CHECK JUST CLEARED"
80 MSG4$="OLD DEPOSIT JUST CREDITED"
90 MSG5$="NEW CHECK/SERVICE CHARGE"
100 MSG6$="NEW DEPOSIT OR INTEREST"
110 TRAP 110:? "ENTER BEGINNING BALANC
E FROM YOUR CHECKBOOK":INPUT YOURBAL
120 TRAP 120:? "ENTER BEGINNING BALANC
E FROM YOUR BANK STATEMENT":INPUT BAL
130 TRAP 40000
140 GOTO 170
150 CLOSE #1:? "PRINTER IS NOT OPERATI
ONAL"
160 ? "PLEASE CHECK CONNECTORS"
170 PERM=0
180 ? "WOULD YOU LIKE A PRINTOUT";:INP
UT A$
190 IF LEN(A$)=0 THEN 180
200 IF A$(1,1)="N" THEN 270
210 IF A$(1,1)<>"Y" THEN 180
220 TRAP 150
```

```
230 LPRINT :REM TEST PRINTER
240 PERM=1
250 LPRINT "YOUR BEGINNING BALANCE IS
$";YOURBAL
260 LPRINT "BANK STATEMENT BEGINNING B
ALANCE IS $";BAL:LPRINT
270 TRAP 270:? :? "CHOOSE ONE OF THE F
OLLOWING:"
280 ? "(1) ";MSG1$
290 ? "(2) ";MSG2$
300 ? "(3) ";MSG3$
310 ? "(4) ";MSG4$
320 ? "(5) ";MSG5$
330 ? "(6) ";MSG6$
340 ? "(7) DONE"
350 ?
360 INPUT N:IF N<1 OR N>7 THEN 270
370 TRAP 40000
380 ON N GOSUB 460,500,540,580,620,750
,880
390 MSG$="NEW CHECKBOOK BALANCE IS ":A
MOUNT=YOURBAL:GOSUB 1040
400 MSG$="NEW BANK STATEMENT BALANCE I
S ":AMOUNT=BAL:GOSUB 1040
410 MSG$="OUTSTANDING CHECKS-DEPOSITS=
":AMOUNT=OUTSTAND:GOSUB 1040
420 IF PERM THEN LPRINT
430 GOTO 270
440 REM NEW DEPOSIT OR INTEREST JUST C
REDITED
450 REM OLD CHECK STILL OUTSTANDING
460 MSG$=MSG1$:GOSUB 1080
470 OUTSTAND=OUTSTAND+AMOUNT
480 RETURN
490 REM OLD DEPOSIT NOT CREDITED
500 MSG$=MSG2$:GOSUB 1080
510 OUTSTAND=OUTSTAND-AMOUNT
520 RETURN
530 REM OLD CHECK JUST CLEARED
540 MSG$=MSG3$:GOSUB 1080
550 BAL=BAL-AMOUNT
560 RETURN
570 REM OLD DEPOSIT JUST CREDITED
580 MSG$=MSG4$:GOSUB 1080
590 BAL=BAL+AMOUNT
```

```
600 RETURN
610 REM NEW CHECK OR SERVICE CHARGE JU
ST CLEARED
620 MSG$=MSG5$:GOSUB 1080
630 YOURBAL=YOURBAL-AMOUNT
640 ? "IS NEW CHECK STILL OUTSTANDING"
;:INPUT A$
650 IF LEN(A$)=0 THEN 640
660 IF A$(1,1)<>"N" THEN 700
670 BAL=BAL-AMOUNT
680 IF PERM THEN LPRINT "CHECK HAS CLE
ARED"
690 RETURN
700 IF A$(1,1)<>"Y" THEN 640
710 OUTSTAND=OUTSTAND+AMOUNT
720 IF PERM THEN LPRINT "CHECK IS STIL
L OUTSTANDING"
730 RETURN
740 REM NEW DEPOSIT OR INTEREST JUST C
REDITED
750 MSG$=MSG6$:GOSUB 1080
760 YOURBAL=YOURBAL+AMOUNT
770 ? "HAS YOUR NEW DEPOSIT BEEN CREDI
TED";:INPUT A$
780 IF LEN(A$)=0 THEN 770
790 IF A$(1,1)<>"Y" THEN 830
800 BAL=BAL+AMOUNT
810 IF PERM THEN LPRINT "DEPOSIT HAS B
EEN CREDITED"
820 RETURN
830 IF A$(1,1)<>"N" THEN 770
840 OUTSTAND=OUTSTAND-AMOUNT
850 IF PERM THEN LPRINT "DEPOSIT HAS N
OT BEEN CREDITED"
860 RETURN
870 REM DONE
880 ? "BANK BALANCE MINUS (OUTSTANDING
 CHECKS-DEPOSITS) SHOULD NOW EQUAL YOU
R CHECKBOOK BALANCE"
890 DIF=YOURBAL-(BAL-OUTSTAND)
900 IF DIF<>0 THEN 950
910 ? "IS $";BAL;" THE ENDING BALANCE
ON YOUR BANK STATEMENT";:INPUT A$
920 IF LEN(A$)=0 THEN 910
930 IF A$(1,1)="Y" THEN ? "CONGRATULAT
```

```
IONS:  YOUR CHECKBOOK BALANCES!":END
940 GOTO 970
950 IF DIF>0 THEN ? "YOUR CHECKBOOK TO
TAL IS $";DIF;" OVER YOUR BANK'S TOTAL
":GOTO 970
960 ? "YOUR CHECKBOOK TOTAL IS $";-DIF
;" UNDER YOUR BANK'S TOTAL"
970 ? "WOULD YOU LIKE TO MAKE CORRECTI
ONS";:INPUT A$
980 IF LEN(A$)=0 THEN 970
990 IF A$(1,1)="N" THEN END
1000 IF A$(1,1)<>"Y" THEN 970
1010 ? "YOU CAN ENTER A NEGATIVE DOLLA
R VALUE TO MAKE A CORRECTION"
1020 RETURN
1030 REM MSG PRINT ROUTINE
1040 ? MSG$;"$";AMOUNT
1050 IF PERM THEN LPRINT MSG$;"$";AMOU
NT
1060 RETURN
1070 REM MSG PRINT/INPUT ROUTINE
1080 TRAP 1080:? "ENTER AMOUNT FOR ";M
SG$:INPUT AMOUNT
1090 TRAP 40000
1100 IF PERM THEN LPRINT MSG$;"$";AMOU
NT
1110 RETURN
```

BUBBLE SORT

This program uses the string comparison operator " < =" that orders strings according to the ATASCII values of the various characters. Since ATARI BASIC does not have arrays of strings, all the strings used in this program are actually substrings of one large string. A bubble sort, though relatively slow if there are a lot of items to be stored, is easy to write, fairly short, and simpler to understand than more complex sorts.

```
10 DIM B$(1)
20 GRAPHICS 0:? :? "STRING SORT":?
30 TRAP 30:? :? "ENTER MAXIMUM STRING
LENGTH":INPUT SLEN:SLEN1=SLEN-1
40 IF SLEN<1 OR INT(SLEN)<>SLEN THEN ?
 "PLEASE ENTER A POSITIVE INTEGER >0":
GOTO 30
50 TRAP 50:? :? "ENTER MAXIMUM NUMBER
OF ENTRIES"
60 ? "(ENTRIES THAT ARE SHORTER THAN T
HE MAXIMUM ARE PADDED WITH BLANKS)"
70 INPUT ENTRIES
80 IF ENTRIES<2 OR INT(ENTRIES)<>ENTRI
ES THEN ? "PLEASE ENTER A POSITIVE INT
EGER >1":GOTO 50
90 TRAP 40000
100 DIM A$(SLEN*ENTRIES),TEMP$(SLEN)
110 ? :? "ENTER STRINGS ONE AT A TIME"
120 ? "ENTER EMPTY STRING WHEN DONE (J
UST HIT RETURN)"
130 ? :? "PLEASE STAND BY WHILE THE ST
RINGS ARE BEING CLEARED..."
140 FOR I=1 TO SLEN*ENTRIES:A$(I,I)="
":NEXT I
150 ? :?
160 I=1
170 FOR J=1 TO ENTRIES
180 ? "#";J;" ";:INPUT TEMP$
190 IF LEN(TEMP$)=0 THEN ENTRIES=J-1:G
OTO 230
200 A$(I,I+SLEN1)=TEMP$
210 I=I+SLEN
220 NEXT J
230 ? :? :? "PLEASE STAND BY WHILE THE
 STRINGS ARE BEING SORTED..."
240 GOSUB 400:REM CALL SORT ROUTINE
250 ? :?
```

```
260 I=1
270 FOR K=1 TO ENTRIES
280 ? "#";K;" ";A$(I,I+SLEN1)
290 I=I+SLEN
300 NEXT K
310 TRAP 310:? :? "WOULD YOU LIKE A PR
INTOUT":INPUT B$
320 IF B$(1,1)="Y" THEN 340
330 END
340 I=1:LPRINT :FOR K=1 TO ENTRIES
350 LPRINT "#";K;" ";A$(I,I+SLEN1)
360 I=I+SLEN:NEXT K:END
370 REM STRING BUBBLE SORT ROUTINE
380 REM INPUT:A$,SLEN,ENTRIES
390 REM TEMP$ MUST HAVE A DIMENSION OF
SLEN
400 MAX=SLEN*(ENTRIES-1)+1
410 FOR I=1 TO MAX STEP SLEN
420 DONE=1
430 FOR K=1 TO MAX-I-SLEN1 STEP SLEN
440 KSLEN1=K+SLEN1:KSLEN=K+SLEN:KSLENS
LEN1=KSLEN+SLEN1
450 IF A$(K,KSLEN1)<=A$(KSLEN,KSLENSLE
N1) THEN GOTO 480
460 DONE=0
470 TEMP$=A$(K,KSLEN1):A$(K,KSLEN1)=A$
(KSLEN,KSLENSLEN1):A$(KSLEN,KSLENSLEN1
)=TEMP$
480 NEXT K
490 IF DONE THEN RETURN
500 NEXT I
510 RETURN
```

LIGHT SHOW

This program demonstrates another aspect of ATARI graphics. It uses graphics mode 7 for high resolution and the PLOT and DRAWTO statements to draw the lines. In line 20, the title will be more effective if it is entered in inverse video (use the ATARI logo key).

```
10 FOR ST=1 TO 8:GRAPHICS 7
20 POKE 752,1
30 ? :? " ATARI LIGHT SHOW "
40 SETCOLOR 2,2,2
50 SETCOLOR 1,2*ST,8:COLOR 2
60 FOR DR=0 TO 80 STEP ST
70 PLOT 0,0:DRAWTO 159,DR
80 NEXT DR
90 FOR DR=159 TO 0 STEP -ST
100 PLOT 0,0:DRAWTO DR,79
110 NEXT DR
120 FOR N=1 TO 3500:NEXT N
130 NEXT ST:GOTO 10
```

UNITED STATES FLAG

This program involves switching colors to set up the stripes. It uses graphics mode 7 plus 16 so that the display appears as a full-screen. Note the correspondence of the COLOR statements with the SETCOLOR statements. For fun and experimentation purposes, add a SOUND statement and use a READ/DATA combination to add "The Star Spangled Banner" after line 400. (Refer to Section 10.)

```
10 REM DRAW THE UNITED STATES FLAG
20 REM HIGH RESOLUTION 4-COLOR GRAPHIC
S, NO TEXT WINDOW
30 GRAPHICS 7+16
40 REM SETCOLOR 0 RELATES TO COLOR 1
50 SETCOLOR 0,4,4:RED=1
60 REM SETCOLOR 1 RELATES TO COLOR 2
70 SETCOLOR 1,0,14:WHITE=2
80 REM SETCOLOR 2 RELATES TO COLOR 3
90 BLUE=3:REM DEFAULTS TO BLUE
100 REM DRAW 13 RED AND WHITE STRIPES
110 C=RED
120 FOR I=0 TO 12
130 COLOR C
140 REM EACH STRIPE HAS SEVERAL HORIZO
```

```
NTAL LINES
150 FOR J=0 TO 6
160 PLOT 0,I*7+J
170 DRAWTO 159,I*7+J
180 NEXT J
190 REM SWITCH COLORS
200 C=C+1:IF C>WHITE THEN C=RED
210 NEXT I
220 REM DRAW BLUE RECTANGLE
230 COLOR BLUE
240 FOR I=0 TO 48
250 PLOT 0,I
260 DRAWTO 79,I
270 NEXT I
280 REM DRAW 9 ROWS OF WHITE STARS
290 COLOR WHITE
300 K=0:REM START WITH ROW OF 6 STARS
310 FOR I=0 TO 8
320 Y=4+I*5
330 FOR J=0 TO 4:REM 5 STARS IN A ROW
340 X=K+5+J*14:GOSUB 480
350 NEXT J
360 IF K<>0 THEN K=0:GOTO 400
370 REM ADD 6TH STAR EVERY OTHER LINE
380 X=5+5*14:GOSUB 480
390 K=7
400 NEXT I
410 REM IF KEY HIT THEN STOP
420 IF PEEK(764)=255 THEN 420
430 REM OPEN TEXT WINDOW WITHOUT CLEAR
ING SCREEN
440 GRAPHICS 7+32
450 REM CHANGE COLORS BACK
460 SETCOLOR 0,4,4:SETCOLOR 1,0,14
470 END
480 REM DRAW 1 STAR CENTERED AT X,Y
490 PLOT X-1,Y:DRAWTO X+1,Y
500 PLOT X,Y-1:PLOT X,Y+1
510 RETURN
520 REM TO ADD A MUSIC ROUTINE,INSERT
A GOSUB AT LINE 405 AND APPEND
530 REM THE MUSIC ROUTINE STATEMENTS A
FTER THIS REM STATEMENT.
```

SEAGULL OVER OCEAN

This program combines graphics and sounds. The sounds are not "pure" sounds, but simulate the roar of the ocean and the gull's "tweet". The graphics symbols used to simulate the gull could not be printed on the line printer. Enter the following characters in line 20.

20 BIRD$ = " v– – "

To get these symbols, use **CTRL** G, **CTRL** F, **CTRL** R, **CTRL** R.

```
10 DIM BIRD$(6)
20 BIRD$="":FLAG=1:ROW=10:COL=10
30 GRAPHICS 1:POKE 756,226:POKE 752,1
40 SETCOLOR 0,0,0:SETCOLOR 1,8,14
50 PRINT #6;"       the ocean"
60 R=INT(RND(0)*11)
70 POSITION 17,17:FOR T=0 TO 10
80 SOUND 0,T,8,4:FOR A=1 TO 50
90 NEXT A:IF RND(0)<0.8 THEN 150
100 PITCH=INT(RND(0)*5)+5
110 FOR D=1 TO 5
120 VOLUME=INT(RND(0)*10)
130 SOUND 1,PITCH+D,10,VOLUME
140 NEXT D:SOUND 1,0,0,0
150 GOSUB 270
160 NEXT T:FOR T=10 TO 0 STEP -1
170 SOUND 0,T,8,4:FOR A=1 TO 50
180 NEXT A:IF RND(0)<0.8 THEN 240
190 PITCH=INT(RND(0)*5)+10
200 FOR D=1 TO 5
210 VOLUME=INT(RND(0)*10)
220 SOUND 1,PITCH-D,10,VOLUME
230 NEXT D:SOUND 1,0,0,0
240 FOR H=1 TO 10:NEXT H
250 GOSUB 270
260 NEXT T:GOTO 60
270 GOSUB 320
280 POSITION COL,ROW
290 PRINT #6;BIRD$(FLAG,FLAG+1)
300 FLAG=FLAG+2:IF FLAG=5 THEN FLAG=1
310 RETURN
320 IF RND(0)>0.5 THEN RETURN
330 POSITION COL,ROW
340 PRINT #6;"  "
350 A=INT(RND(0)*3)-1
360 B=INT(RND(0)*3)-1
370 ROW=ROW+A:IF ROW=0 THEN ROW=1
```

```
380 IF ROW=20 THEN ROW=19:COL=COL+B
390 IF COL=0 THEN COL=1
400 IF COL>18 THEN COL=18
410 RETURN
```

VIDEO GRAFFITTI

This program requires a Joystick Controller for each player. Each joystick has one color associated with it. By maneuvering the joystick, different patterns are created on the screen. Note the use of the **STICK** and **STRIG** commands.

```
10 GRAPHICS 0
20 ? "VIDEO GRAFFITI"
30 REM X&Y ARRAYS HOLD COORDINATES
40 REM FOR UP TO 4 PLAYER POSITIONS
50 REM COLR ARRAY HOLDS COLORS
60 DIM A$(1),X(3),Y(3),COLR(3)
70 ? "USE JOYSTICKS TO DRAW PICTURES"
80 ? "PRESS BUTTONS TO CHANGE COLORS"
90 ? "INITIAL COLORS:"
100 ? "JOYSTICK 1 IS RED"
110 ? "JOYSTICK 2 IS WHITE"
120 ? "JOYSTICK 3 IS BLUE"
130 ? "JOYSTICK 4 IS BLACK-BACKGROUND"
140 ? "BLACK LOCATION IS INDICATED BY
 BRIEF FLASH OF RED"
150 ? "IN GRAPHICS 8,JOYSTICKS 1 AND 3
 ARE WHITE AND 4 IS BLUE"
160 ? "HOW MANY PLAYERS (1-4)";
170 INPUT A$:IF LEN(A$)=0 THEN A$="1"
180 JOYMAX=VAL(A$)-1
190 IF JOYMAX<0 OR JOYMAX>=4 THEN 160
200 ? "GRAPHICS3(40X24),5(80X48)"
210 ? "7(160X96),OR 8(320X192)";
220 INPUT A$:IF LEN(A$)=0 THEN A$="3"
230 A=VAL(A$)
240 IF A=3 THEN XMAX=40:YMAX=24:GOTO 2
90
250 IF A=5 THEN XMAX=80:YMAX=48:GOTO 2
90
260 IF A=7 THEN XMAX=160:YMAX=96:GOTO
290
270 IF A=8 THEN XMAX=320:YMAX=192:GOTO
 290
280 GOTO 147:REM A NOT VALID
```

```
290 GRAPHICS A+16
300 FOR I=0 TO JOYMAX:X(I)=XMAX/2+I:Y(
I)=YMAX/2+I:NEXT I:REM START NEAR CENT
ER OF SCREEN
310 IF A<>8 THEN 350
320 FOR I=0 TO 2:COLR(I)=1:NEXT I
330 SETCOLOR 1,9,14:REM LT. BLUE
340 GOTO 380
350 FOR I=0 TO 2:COLR(I)=I+1:NEXT I
360 SETCOLOR 0,4,6:REM RED
370 SETCOLOR 1,0,14:REM WHITE
380 COLR(3)=0
390 FOR J=0 TO 3
400 FOR I=0 TO JOYMAX:REM CHECK JOYSTI
CKS
410 REM CHECK TRIGGER
420 IF STRIG(I) THEN 470
430 IF A<>8 THEN 460
440 COLR(I)=COLR(I)+1:IF COLR(I)=2 THE
N COLR(I)=0:REM TWO COLOR MODE
450 GOTO 470
460 COLR(I)=COLR(I)+1:IF COLR(I)>=4 TH
EN COLR(I)=0:REM FOUR COLOR MODE
470 IF J>0 THEN COLOR COLR(I):GOTO 500
480 IF COLR(I)=0 THEN COLOR 1:GOTO 500
490 COLOR 0:REM BLINK CURRENT SQUARE O
N AND OFF
500 PLOT X(I),Y(I)
510 JOYIN=STICK(I):REM READ JOYSTICK
520 IF JOYIN=15 THEN 690:REM NO MOVEME
NT
530 COLOR COLR(I):REM MAKE SURE COLOR
IS ON
540 PLOT X(I),Y(I)
550 IF JOYIN>8 THEN 600
560 X(I)=X(I)+1:REM MOVE RIGHT
570 REM IF OUT OF RANGE THEN WRAPAROUN
D
580 IF X(I)>=XMAX THEN X(I)=0
590 GOTO 630
600 IF JOYIN>=12 THEN 630
610 X(I)=X(I)-1:REM MOVE LEFT
620 IF X(I)<0 THEN X(I)=XMAX-1
630 IF JOYIN<>5 AND JOYIN<>9 AND JOYIN
<>13 THEN 660
```

```
640 Y(I)=Y(I)+1:IF Y(I)>=YMAX THEN Y(I
)=0:REM MOVE DOWN
650 GOTO 680
660 IF JOYIN<>6 AND JOYIN<>10 AND JOYI
N<>14 THEN 680
670 Y(I)=Y(I)-1:IF Y(I)<0 THEN Y(I)=YM
AX-1:REM MOVE UP
680 PLOT X(I),Y(I)
690 NEXT I
700 NEXT J
710 GOTO 390
```

KEYBOARD CONTROLLER

This program alters registers on a chip called a PIA. To set these back to the default values in order to do further I/O, hit **SYSTEM RESET** or **POKE** PACTL,60. If this program is to be loaded from disk, use **LOAD**, not **RUN** and wait for the busy light on the disk drive to go out. Do not execute the program *before* this light goes out, otherwise the disk will continue to spin.

```
10 GRAPHICS 0
20 ? :? "KEYBOARD CONTROLLER DEMO"
30 DIM ROW(3),I$(13),BUTTON$(1)
40 GOSUB 100
50 FOR CNT=1 TO 4
60 POSITION 2,CNT*2+5:? "CONTROLLER #"
;CNT;":"
70 NEXT CNT
80 FOR CNT=1 TO 4:GOSUB 170:POSITION 1
9,CNT+CNT+5:? BUTTON$:NEXT CNT
90 GOTO 80
100 REM ** SET UP FOR CONTROLLERS**
110 PORTA=54016:PORTB=54017:PACTL=5401
8:PBCTL=54019
120 POKE PACTL,48:POKE PORTA,255:POKE
PACTL,52:POKE PORTA,221
130 POKE PBCTL,48:POKE PORTB,255:POKE
PBCTL,52:POKE PORTB,221
140 ROW(0)=238:ROW(1)=221:ROW(2)=187:R
OW(3)=119
150 I$="  123456789*0#"
160 RETURN
170 REM **RETURN BUTTON$ WITH CHARACTE
R FOR BUTTON WHICH HAS BEEN PRESSED ON
 CONTROLLER CNT(1-4)**
180 REM **NOTE: A 1 IS RETURNED IF NO
```

```
CONTROLLER IS CONNECTED**
190 REM **A SPACE IS RETURNED IF THE C
ONTROLLER IS CONNECTED BUT NO KEY HAS
BEEN PRESSED**
200 PORT=PORTA:IF CNT>2 THEN PORT=PORT
B
210 P=1
220 PAO=CNT+CNT-2
230 FOR J=0 TO 3
240 POKE PORT,ROW(J)
250 FOR I=1 TO 10:NEXT I
260 IF PADDLE(PAO+1)>10 THEN P=J+J+J+2
:GOTO 300
270 IF PADDLE(PAO)>10 THEN P=J+J+J+3:G
OTO 300
280 IF STRIG(CNT-1)=0 THEN P=J+J+J+4:G
OTO 300
290 NEXT J
300 BUTTON$=I$(P,P)
310 RETURN
```

TYPE-A-TUNE

This program assigns musical note values to the keys on the top row of the keyboard. Press only one key at a time.

KEY	MUSICAL VALUE
INSERT	B
CLEAR	B♭ (or A#)
0	A
9	A♭ (or G#)
8	G
7	F# (or G♭)
6	F
5	E
4	E♭ (or D#)
3	D
2	D♭ (or C#)
1	C

```
10 DIM PITCH(12),TUNE(12):COLOR 1
20 GRAPHICS 0:? :? "TYPE-A-TUNE"
30 ? :? "PRESS KEYS 1-9,0,<,> TO PRODU
CE NOTES"
40 ? "RELEASE ONE KEY BEFORE PRESSING
THE NEXT"
```

```
50 ? "OTHERWISE THERE MAY BE A DELAY"
60 FOR X=1 TO 12:READ A:PITCH(X)=A:NEX
T X
70 FOR X=1 TO 12:READ A:TUNE(X)=A:NEXT
 X
80 OPEN #1,4,0,"K:"
90 OLDCHR=-1
100 A=PEEK(764):IF A=255 THEN 100
110 IF A=OLDCHR THEN 150
120 OLDCHR=A
130 FOR X=1 TO 12:IF TUNE(X)=A THEN SO
UND 0,PITCH(X),10,8:GOTO 150
140 NEXT X
150 I=INT(PEEK(53775)/4):IF (I/2)=INT(
I/2) THEN 100
160 POKE 764,255:SOUND 0,0,0,0:OLDCHR=
-1:GOTO 100
170 DATA 243,230,217,204,193,182,173,1
62,153,144,136,128
180 DATA 31,30,26,24,29,27,51,53,48,50
,54,55
```

To play "Mary Had A Little Lamb" press the following keys:

5, 3, 1, 3, 5, 5, 5 3, 3, 3 5, 8, 8 5, 3, 1, 3, 5, 5 5, 5, 3, 3, 5, 3, 1

COMPUTER BLUES

This program generates random musical notes to "write" some very interesting melodies for the programmed bass.

```
10 GRAPHICS 1
20 ? "BASS TEMPO--SELECT A NUMBER."
30 ? "(FASTEST TEMPO=1)"
40 PTR=1:THNOT=1:CHORD=1
50 PRINT "PRESS RETURN"
60 INPUT TEMPO
70 GRAPHICS 2+16:GOSUB 630
80 DIM BASE(3,4)
90 DIM LOW(3)
100 DIM LINE(16)
110 DIM JAM(3,7)
120 FOR X=1 TO 3
```

```
130 FOR Y=1 TO 4
140 READ A:BASE(X,Y)=A
150 NEXT Y
160 NEXT X
170 FOR X=1 TO 3:READ A:LOW(X)=A
180 NEXT X
190 FOR X=1 TO 16:READ A:LINE(X)=A:NEX
T X
200 FOR X=1 TO 3
210 FOR Y=1 TO 7
220 READ A:JAM(X,Y)=A:NEXT Y:NEXT X
230 GOSUB 370
240 T=T+1
250 GOSUB 270
260 GOTO 230
270 REM PROCESS HIGH STUFF
280 IF RND(0)<0.25 THEN RETURN
290 IF RND(0)<0.5 THEN 330
300 NT=NT+1
310 IF NT>7 THEN NT=7
320 GOTO 350
330 NT=NT-1
340 IF NT<1 THEN NT=1
350 SOUND 2,JAM(CHORD,NT),10,NT*2
360 RETURN
370 REM PROCESS BASE STUFF
380 IF BASS=1 THEN 450
390 BOUR=BOUR+1
400 IF BOUR<>TEMPO THEN 420
410 BASS=1:BOUR=0
420 SOUND 0,LOW(CHORD),10,4
430 SOUND 1,BASE(CHORD,THNOT),10,4
440 RETURN
450 SOUND 0,0,0,0
460 SOUND 1,0,0,0
470 BOUR=BOUR+1
480 IF BOUR<>1 THEN 560
490 BOUR=0:BASS=0
500 THNOT=THNOT+1
510 IF THNOT<>5 THEN 560
520 THNOT=1
530 PTR=PTR+1
540 IF PTR=17 THEN PTR=1
550 CHORD=LINE(PTR)
560 RETURN
```

```
570 DATA 162,144,136,144,121,108,102,1
08,108,96,91,96
580 DATA 243,182,162
590 DATA 1,1,1,1,2,2,2,2,1,1,1,1,3,2,1
,1
600 DATA 60,50,47,42,40,33,29
610 DATA 60,50,45,42,40,33,29
620 DATA 81,68,64,57,53,45,40
630 PRINT #6:PRINT #6:PRINT #6
640 PRINT #6;"   COMPUTER BLUES"
650 PRINT #6:PRINT #6
660 RETURN
```

DECIMAL/HEXADECIMAL CONVERSION PROGRAM

This program can be typed in and used to convert hexadecimal numbers to decimal numbers and vice versa.

```
10 DIM A$(9),AD$(1)
20 GRAPHICS 0:? :? "HEX CONVERSION"
30 ? "ENTER'D'FOR DEC TO HEX CONVERT"
40 ? "ENTER'H'FOR HEX TO DEC CONVERT"
50 INPUT A$
60 IF LEN(A$)=0 THEN 30
70 IF A$="H" THEN 300
80 IF A$<>"D" THEN 30
90 TRAP 90
100 ? "ENTER A DECIMAL NUMBER"
110 ? "DEC:";:INPUT N
120 IF N<0 OR N>=1E+10 OR N<>INT(N) TH
EN GOTO 100
130 I=9
140 TEMP=N:N=INT(N/16)
150 TEMP=TEMP-N*16
160 IF TEMP<10 THEN A$(I,I)=STR$(TEMP)
:GOTO 180
170 A$(I,I)=CHR$(TEMP-10+ASC("A"))
180 IF N<>0 THEN I=I-1:GOTO 140
190 ? "HEX: ";A$(I,9):? :GOTO 110
300 TRAP 300
310 ? :? "ENTER A HEX NUMBER"
320 ? "HEX:";:INPUT A$:N=0
330 IF LEN(A$)=0 THEN 300
340 FOR I=1 TO LEN(A$)
350 AD$=A$(I,I):IF AD$<"0" THEN 300
```

```
360 IF AD$>"9" THEN GOTO 380
370 N=N*16+VAL(AD$):GOTO 410
380 IF AD$<"A" THEN 300
390 IF AD$>"F" THEN 300
400 N=N*16+ASC(AD$)-ASC("A")+10
410 NEXT I
420 ? "DEC: ";N:? :GOTO 320
```

MEMORY LOCATIONS

Note: Many of these locations are of primary interest to expert programmers and are included here as a convenience. The labels given are used by ATARI programmers to make programs more readable.

LABEL	DECIMAL LOCATION	HEXADECIMAL LOCATION	COMMENTS AND DESCRIPTION
APPMHI	14,15	D, E	Highest location used by BASIC (LSB, MSB)
RTCLOK	18,19,20	12,13,14	TV frame counter (1/60 sec.) (LSB, NSB, MSB). Time in seconds = (**PEEK**(18) + **PEEK** (19)*256 + **PEEK**(20)*256*256)/60
SOUNDR	65	41	Noisy I/O Flag (0 = quiet)
	77		Attract Mode Flag (128 = Attract mode)
LMARGIN, RMARGIN	82,83	52,53	Left, Right Margin (Defaults 2, 39)
ROWCRS	84	54	Current cursor row (graphics window).
COLCRS	85,86	55,56	Current cursor column (graphics window).
OLDROW	90	5A	Previous cursor row (graphics window).
OLDCOL	91,92	5B	Previous cursor column (graphics window).
	93	5C	Data under cursor (graphics window unless mode 0).
RAMTOP	106	6A	Actual top of memory (number of pages).
LOMEM	128,129	80,81	BASIC low memory pointer.
MEMTOP	144,145	90,91	BASIC top of memory pointer.
STOPLN	186,187	BA,BB	Line number at which **STOP** or **TRAP** occurred (2-byte binary number).
ERRSAV	195	C3	Error number.
PTABW	201	C9	Print tab width (defaults to 10)
FR0	212,213	D4,D5	Low and high bytes of value to be returned to BASIC from USR function.
RADFLG	251	FB	RAD/DEG flag (0 = radians, 6 = degrees).
LPENH	564	234	Light Pen* Horizontal value.
LPENV	565	235	Light Pen* Vertical value.
TXTROW	656	290	Cursor row (text window)
TXTCOL	657,658	291,292	Cursor column (text window)
COLOR0	708	2C4	Color Register 0
COLOR1	709	2C5	Color Register 1
COLOR2	710	2C6	Color Register 2

*Future product.

LABEL	DECIMAL LOCATION	HEXADECIMAL LOCATION	COMMENTS AND DESCRIPTIONS
COLOR3	711	2C7	Color Register 3
COLOR4	712	2C8	Color Register 4
MEMTOP	741,742	2E5,2E6	OS top of available user memory pointer (LSB, MSB)
MEMLO	743,744	2E7,2E8	OS low memory pointer
CRSINH	752	2F0	Cursor inhibit (0 = cursor on, 1 = cursor off)
CHACT	755	2F3	Character mode register (4 = vertical reflect; 2 = normal; 1 = blank)
CHBAS	756	2F4	Character base register (defaults to 224) (224 = upper case, 226 = lower case characters)
ATACHR	763	2FB	Last ATASCII character.
CH	764	2FC	Last keyboard key pressed; internal code; (255 clears character).
FILDAT	765	2FD	Fill data for graphics Fill (XIO).
DSPFLG	766	2FE	Display Flag (1 = display control character).
SSFLAG	767	2FF	Start/Stop flag for paging (0 = normal listing) Set by **CTRL** 1.
HATABS	794	31A	Handler address table (3 bytes/handler)
IOCB	832	340	I/O control blocks (16 bytes/IOCB)
	1664-1791	680-6FE	Spare RAM
CONSOL	53279	D01F	Console switches (bit 2 = Option; bit 1 = Select; bit 0 = Start. **POKE** 53279, 0 before reading. 0 = switch pressed.)
PORTA PORTB	54016 54017	D300 D301	PIA Port A Controller Jack I/O ports. PIA Port B Initialized to hex 3C.
PACTL	54018	D302	Port A Control Register (on Program Recorder 52 = ON, 60 = OFF).
PBCTL	54019	D303	Port B control register.
SKCTL	53775	D20F	Serial Port control register. Bit 2 = 0 (last key still pressed).
SAVMSC	88,89 123,184 182	58,59	Points to screen data area. **Read/data** pointer (line #). **Read** (displacement in line).
SDLIST	560,561 580 694 53770	230,231	Display list pointer. Coldstart flag. Inverse video (128 = on, 0 = off) Random # between 0 and 255.
POKMSC	16/53774		**Poke** both w/64 to disable BREAK key (reenabled when entering new graphics mode).

J APPENDIX TABLE OF MODES AND SCREEN FORMATS

TABLE 9-1 TABLE OF MODES AND SCREEN FORMATS

SCREEN FORMAT							
Graphics Mode	Mode Type	Columns	Rows—Split Screen**	Rows—Full Screen	Number of Colors	RAM Required (Bytes) Split	Full
0	TEXT	40	—	24	1-1/2		992
1	TEXT	20	20	24	5	674	672
2	TEXT	20	10	12	5	424	420
3	GRAPHICS	40	20	24	4	434	432
4	GRAPHICS	80	40	48	2	694	696
5	GRAPHICS	80	40	48	4	1174	1176
6	GRAPHICS	160	80	96	2	2174	2184
7	GRAPHICS	160	80	96	4	4190	4200
8	GRAPHICS	320	160	192	1-1/2	8112	8138
9*	GRAPHICS	80	—	192	1		8138
10*	GRAPHICS	80	—	192	9		8138
11*	GRAPHICS	80	—	192	16		8138
12***	GRAPHICS	40	20	24	5	1154	1152
13***	GRAPHICS	40	10	12	5	664	660
14***	GRAPHICS	160	160	192	2	4270	4296
15***	GRAPHICS	160	160	192	4	8112	8138

*GTIA Mode Only
**Refer to Figure 9-1.
***1200XL Only

INDEX

A
Abbreviations, 4-5
  Commands in headings, 4
ABS, 40
adata, 5
ADR, 42, 76
aexp, 4
aop, 4
Array, 3-4 49
ASC, 45
ATN, 42
Audio track of cassette, 29
avar, 45

B
BASIC, 1
Blanks (*see* Spaces)
Booting DOS, 31
Braces, 4
Brackets, 4
Branching,
  Conditional Statements, 22
  Unconditional Statements, 21
Brightness (*see* Luminance)
Bubble Sort Program, H-5
Buzzer, 16
  Deferred Mode, F-1
  Direct Mode, 16
BYE, 12

C
C-Scale Program, 67
Central Input/output Subsystem, 29
Character
  Assigning Color to, 63
  ATASCII, C-1 through C-8
  Display at specified locations, 53, 54
  Set, internal, 62
  Sizes in Text modes, 53
Chaining Programs, 38
CHR$, 45
CIO (*see* Central Input/output Subsystem)
CLEAR key, 7
Clear Screen,
  Deferred mode, 7, 16, 53
  Direct mode, 6, 53
CLOAD, 30
CLOG, 40
CLOSE, 33
CLR, 51
Codes,
  Device, 29-30
Colons, 4, 70
COLOR, 53
Color
  Assigning, 63
  Changing, 58
  Default, 54, 59
  Registers, 58
COM (*see* DIM)
cmdno, 37
Comma, 32-33
Command Strings, 1
Commands
  BYE, 12
  CONT, 12
  END, 13
  LET, 13
  LIST, 13
  NEW, 13
  REM, 13
  RUN, 13
  STOP, 13
Concatenation, String, 47
Conservation,
  Memory, 70
Constant, 2
CONT, 12
Control Key, 15-17
Controllers,
  Game, 68
COS, 42
CSAVE, 30
Cursor, 12
  Graphics, 56
  Inhibit, 53

D
Default
  colors, 53
  disk drive, 30, 36
  margins in Mode 0, 53
  tab settings, 7
Deferred mode, 7
DEG, 42
Devices, 29-30
Delete line, 16
DIM, 50
Direct mode, 6
Disk Drive
  Default number, 30, 36
  Requirements (*see ATARI DOS Manual*)
Disk file
  Modification of BASIC program, 38
Display, split-screen override, 52, 54
Distortion, 66
DOS, 31
Double-Key Functions, 16
DRAWTO, 55

E
Editing, screen, 15
Editor, Screen, 30
END, 12, 13
End of file, 20
Enter, 31
Error messages, B-1 through B-3, 11
Escape key, 6
  with Control Graphics Symbols, F-1
EXP, 40
exp, 9
Exponentiation symbol, 9
Expression, 1
  Arithmetic (*see* aexp)
  Logical (*see* lexp)
  String (*see* sexp)

F
filename, breakdown, 34
filespec, 5
  Usage, 33, 34
Fill (XIO), 61
FOR/NEXT, 18
building arrays and matrices, 51
  with STEP, 18
  without STEP, 18
FRE, 35
Function, 2
  Arithmetic
    ABS, 40
    CLOG, 40
    EXP, 40
    INT, 41
    LOG, 41

RND, 41
SGN, 41
SQR, 41
Built-in, 9
Derived, E-1
Library, 40
Special Purpose, 42
ADR, 42
FRE, 42
PEEK, 43
POKE, 43
USR, 43
Trigonometric, 42
ATN, 42
COS, 42
DEG, 42
RAD, 42
SIN, 42

G
Game controllers
Joystick, 68
Paddle, 68
Video Graffitti program, H-12 through H-13
Game controller commands
PADDLE, 68
PTRIG, 69
STICK, 69
STRIG, 69
GET, 35, 57
GOSUB/RETURN, 19, 24, 26
GOTO, 21
with conditional branching, 21
GRAPHICS, 52
Graphics
Modes, 52
Statements, 56
COLOR, 56
DRAWTO, 56
GET, 57
GRAPHICS, 56
LOCATE, 56
PLOT, 57
POSITION, 57
PUT, 57
SETCOLOR, 58
XIO (Fill), 61
Graphics Control Characters, 65

H
Harmony, 66
Hexcode Loader program, 72-73

I
If/then, 22
INPUT, 31
Input/Output Commands, 29
CLOAD, 30
CLOSE, 33-34
CSAVE, 30
DATA, 35
DOS, 31
ENTER, 31
GET, 35
INPUT, 31
LOAD, 32
LPRINT, 32
NOTE, 33
OPEN, 33
POINT, 34
PRINT, 4, 6
PUT, 35
READ, 35
SAVE, 36
STATUS, 36
XIO, 37
Input/Output Devices
Disk Drives (D:), 30
Keyboard (K:), 29
Line Printer (L:) 29
Program Recorder (C:), 29
RS-232 Interface (R:) 30
Screen Editor (E:), 30
TV Monitor (S:), 30
INT, 41
Internal pointer
for DATA, 27
Input/Output Control Block, 29
Inverse Key, 5
Invisible graphics cursor, 56
IOCB (*see* Input/Output Control Block)

J
Joystick Controller, 68

K
Keyboard (K:), 29
Keys
Special Function
ATARI, 6
BACKSPACE, 7
BREAK, 6
CAPS, 6
CAPS/LOWER, 6
CLEAR, 7
DELETE, 7
ESCAPE, 6
INSERT, 7
RETURN, 7
SYSTEM RESET, 6
TAB, 7
Editing
CTRL (Control) Key, 15
SHIFT key, 15
Cursor Control 16
Down arrow, 16
Left arrow, 16
Right arrow, 16
Up arrow, 16
Keywords
BASIC, A-1 through A-5

L
LEN, 46
LET, 2, 4, 13
Letters
Capital (upper case), 4
Lower case, 4
lexp, 5
Light Show Program, H-8
Line
Format, 4
Logical, 2
Numbers, 4
Physical, 2
lineno, 5
LIST, 13
LOAD, 32
Load program from cassette tape, 30
LOCATE, 56
LOG, 41
Loops
Endless, 20
Nested, 18
lop, 5
LPRINT, 33
before CSAVE, 31
Luminance, 60

M
Mandatory # symbol, 33
Margins
Changing, 43
Default in mode 0, 54
Matrix, 49-51
Variable, 4
Memory Map, D-1
Modes, graphics, 54, 55
Modes, operating
Deferred, 6
Direct, 6
Execute, 6

Memo Pad, 6, 31
Modes, text, 54
Override split-screen, 54
Multiple commands (*see* Command Strings)
mvar, 4

N
NEW, 14
Notations
floating point, 47
in manual, 3
Note, 33

O
ON/GOSUB, 24
ON/GOTO, 24
OPEN, 33-34
Operating Modes, 6
Operators, 2
Arithmetic, 9, 10
Binary, 9, 10
Logical, 9
Relational, 9
Unary, 9
Output devices, 29-30
Oversized programs (*see* Chaining Programs)

P
Paddle Controller, 68
Parentheses,
Usage, 10, 71
PEEK, 43
Peripheral devices (*see* Input/Output Devices)
Pitch
Definition, 66
Values, 66
Pixel, 57
Size in modes, 56
PLA, 71
PLOT, 57
POINT, 34
POKE, 43
POP, 25, 26
POSITION, 57
Precedence, operator, 8
PRINT, 33, 35
Printer listing, 13
Program continuation, 14
Programs,
Machine language, 71
User, Appendix H
PUT, 35, 57

Q
Question mark as prompt, 31
Quotation marks, 2

R
RAD, 42
RAM (Random Access Memory), 29
Random Access to disk file, 34
READ, 35
Direct mode, 36
REM, 14
RESTORE, 27
RETURN Key, 6
Return, Abnormal (*see* POP)
Rollover,
Keyboard, 11
RND, 41
RS-232(R:), 29
RTS, 63
RUN, 1

S
SAVE, 36
Save programs on cassette tape, 30
Screen Display (*see* TV Monitor)
Screen Edit, 15
Screen Editor (E:), 30
Seagull Over Ocean Program, H-11
Self Test, 8
Semicolon, 28
SETCOLOR, 60
sexp, 5
SGN, 41
Shift Key, 15-17
SIN, 42
SOUND, 60
terminating, 60
Spaces, 70
Special Function Keys, 16
SQR, 41
Stack, 19
GOSUB, 19
Hardware, 43
loop addresses, 19, 24
POP, 25
Statement,
Program, 18
FOR, 18
GOSUB, 19, 24, 26
GOTO, 21
IF, 22
ON/GOSUB, 24
ON/GOTO, 24
POP, 25
RESTORE, 27
RETURN, 19, 24
STEP, 18
THEN, 22
TO, 18
TRAP, 28
STATUS, 36
STEP, 18
STOP, 14
String
Comparison, 48
Concatenation, 47
Dimensioning, 45
Functions
ASC, 45
CHR$, 45
LEN, 46
STR$, 46
VAL, 46
Manipulation, 47
Sort, 48
Splitting, 47
Variable, 4
STR$, 46
Subroutine
Definition, 20
GOSUB, 19, 24, 26
Usage, 24
svar, 4

T
Terminology, 1
Text modes, 54
Text Modes Characters Program, H-7
Tokenized version, 3, 30
Tone, clipped, 66
Trigonometric, 47
TRAP, 28
Type-A-Tune Program, H-15

U
Untokenized version, 3
USR, 43

V
VAL, 46
var, 4
Variable, 2
avoiding name limit, 2
Video Graffitti Program, H-12

Volume control, 66
Voice, 66

W
Window
  Graphics, 54
  Text, 54
Wraparound, 11

X
X-coordinate, 54
XIO, 37
XIO Drawline, 61
XIO (Fill), 61

Y
Y-coordinate, 54

Z
Zero
  as Dummy Variable, 38, 42

A Warner Communications Company

© 1983 Atari, Inc. All Rights Reserved.

PRINTED IN U.S.A.
C061456 REV. A BX4211